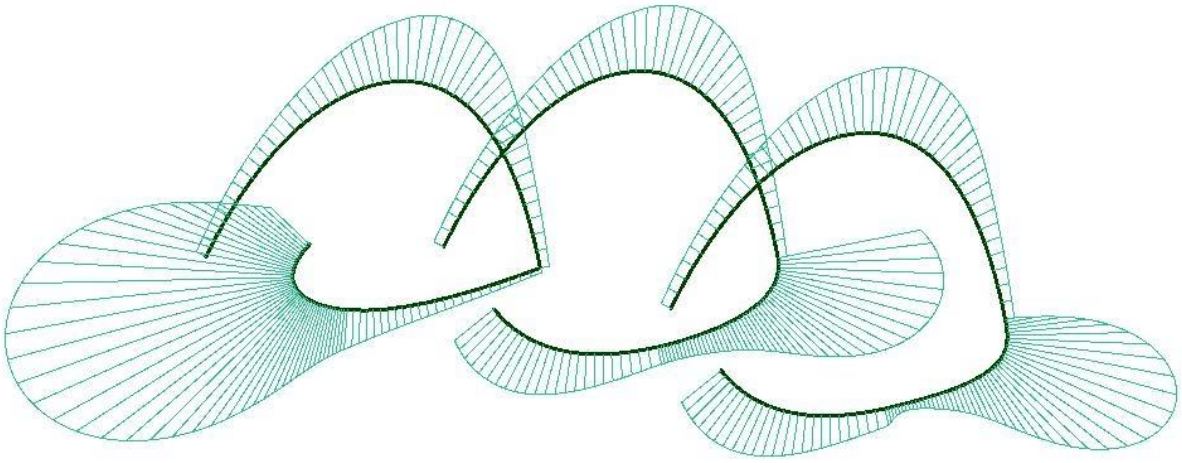


Matemáticas Esenciales

para Diseño Computacional - Cuarta Edición

Rajaa Issa
Robert McNeel & Associates

Matemáticas Esenciales para Diseño Computacional, Cuarta Edición, de Robert McNeel & Associates, 2019, tiene una licencia de Creative Commons Attribution-Share Alike 3.0 en Estados Unidos.

Traducción al español: Noemí Fluixà (noemi@mcneel.com) con la colaboración de Jose Luis y Fernando García del Castillo López.

Introducción

Matemáticas Esenciales para Diseño Computacional presenta los conceptos matemáticos básicos que son necesarios para el desarrollo de métodos computacionales para el modelado en 3D y el diseño gráfico por ordenador. Este documento no es una fuente completa y exhaustiva de información, sino más bien una introducción a los conceptos básicos más usados.

El material está enfocado a diseñadores que tienen poca o ninguna experiencia en matemáticas más allá de la escuela secundaria. Todos los conceptos se explican visualmente usando Grasshopper® (GH), el entorno de modelado generativo para Rhinoceros® (Rhino). Para obtener más información, visite www.rhino3d.com.

El contenido se divide en tres capítulos. En el capítulo 1 se tratan las matemáticas vectoriales, incluida la representación de vectores, las operaciones vectoriales y las ecuaciones de líneas y planos. En el capítulo 2 se estudian las operaciones y transformaciones matriciales. En el capítulo 3 se realiza una revisión en profundidad de las curvas paramétricas, con especial énfasis en las curvas NURBS y los conceptos de continuidad y curvatura. También se hace un repaso general de las superficies y polisuperficies NURBS.

Me gustaría agradecer la excelente y completa revisión técnica del Dr. Dale Lear de Robert McNeel & Associates. Sus valiosos comentarios fueron fundamentales para producir esta edición. También quiero agradecer a la Sra. Margaret Becker, de Robert McNeel & Associates, por revisar la redacción técnica y el formato del documento.

Rajaa Issa
Robert McNeel & Associates

Contenido

1 Matemáticas vectoriales **7**
Representación de un vector 7
Vector de posición 8
Vectores y puntos 8
Longitud vectorial 9
Vector unitario 9
Operaciones vectoriales 10
Operación escalar de vectores 10
Suma de vectores 10
Resta de vectores 11
Propiedades de los vectores 12
Producto escalar de vectores 13
Producto escalar de vectores, longitudes y ángulos 14
Propiedades del producto escalar 15
Producto vectorial 15
Producto vectorial y ángulo entre vectores 16
Propiedades del producto vectorial 17
Ecuación vectorial de una línea 17
Ecuación vectorial de un plano 19
Tutoriales 20
Dirección de cara 20
Entrada: 20
Parámetros: 21
Solución: 21
Caja descompuesta 25
Entrada: 25
Parámetros: 25
Solución: 26
Esferas tangentes 31
Entrada: 31
Parámetros: 31
Solución: 32
2 Matrices y transformaciones **35**
Operaciones con matrices 35
Multiplicación de matrices 35

Método 1 35
Método 2 35
Matriz de identidad 36
Operaciones de transformación 37
Traslación (mover) 37
Rotación 38
Escalado 40
Sesgado 41
Reflejo o espejo 42
Proyección plana 43
Tutorial 43
Transformaciones múltiples 43
Entrada: 44
Entrada adicional: 44
Solución: 45
3 Curvas y superficies paramétricas 49
Curvas paramétricas 50
Parámetro de curva 50
Dominio de la curva o intervalo 51
Cálculo de curvas 52
Vector tangente a una curva 54
Curvas polinomiales cúbicas 54
Cálculo de curvas de Bézier cúbicas 56
Curvas NURBS 56
Grado 57
Puntos de control 57
Peso de los puntos de control 58
Nodos 59
Los nodos son valores de parámetros 59
Multiplicidad de nodos 60
Nodos de multiplicidad total 61
Nodos uniformes 61
Nodos no uniformes 62
Regla de cálculo 64
Características de las curvas NURBS 65
Curvas NURBS ancladas vs. periódicas 66
Pesos 67

Cálculo de curvas NURBS 68
Solución: 69
Continuidad geométrica de curva 70
Curvatura de curva 70
Superficies paramétricas 71
Parámetros de superficie 71
Dominio de superficie 73
Cálculo de superficies 74
Plano tangente de una superficie 74
Continuidad geométrica de superficie 75
Curvatura de superficie 76
Curvaturas principales 76
Curvatura gaussiana 77
Curvatura media 78
Superficies NURBS 78
Características de las superficies NURBS 80
Singularidad en las superficies NURBS 82
Superficies NURBS recortadas 83
Polisuperficies 83
Tutoriales 86
Continuidad entre curvas 86
Entrada: 86
Parámetros: 86
Solución: 86
Superficies con singularidad 92
Entrada: 92
Parámetros: 92
Solución: 92
Referencias **95**
Notas 95

1 Matemáticas vectoriales

Un *vector* representa un parámetro, como la velocidad o la fuerza, que tiene *dirección* y *longitud*. Los vectores en sistemas de coordenadas 3D se representan con un conjunto ordenado de tres números reales de la forma:

v = <a1, a2, a3>

Representación de un vector

En este documento, las letras minúsculas en negrita representarán los vectores. Los componentes de un vector también se representarán entre paréntesis angulares. Las letras mayúsculas representarán los puntos. Las coordenadas de los puntos siempre se representarán entre paréntesis.

Utilizando un sistema de coordenadas y cualquier conjunto de puntos de anclaje en ese sistema, podemos representar o visualizar estos vectores dibujándolos como el segmento de una línea. Dibujamos también la punta de una flecha para representar la dirección de los vectores.

Por ejemplo, si tenemos un vector que tiene una dirección paralela al eje x de un sistema de coordenadas dado y una magnitud de 5 unidades, podemos escribir el vector de la siguiente manera:

v = <5, 0, 0>

Para representar ese vector, necesitamos un punto de anclaje en el sistema de coordenadas. Por ejemplo, todas las flechas de la siguiente figura son representaciones iguales del mismo vector a pesar de que están ancladas en diferentes ubicaciones.

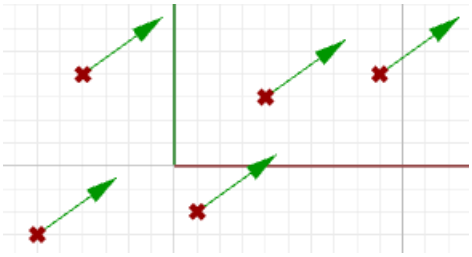


Figura (1): Representación de vectores en el sistema de coordenadas 3D.

> Dado un vector 3D ***v*** = <a1, a2, a3>, todos los componentes del vector a1, a2, a3 son números reales. Además, todos los segmentos de línea desde un punto A(x,y,z) hasta otro punto B(x+a1, y+a2, z+a3) son representaciones equivalentes del vector ***v***.

Entonces, ¿cómo definimos los puntos finales de un segmento de línea que representa un vector dado?

Definamos un punto de anclaje (A) de modo que:

A = (1, 2, 3)

Y un vector:

v = <5, 6, 7>

El punto extremo (B) del vector se calcula sumando los componentes correspondientes del punto de anclaje y el vector **v**:

B = A + **v**

B = (1+5, 2+6, 3+7)

B = (6, 8, 10)

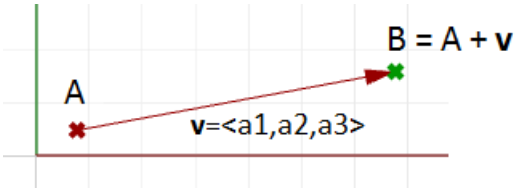


Figura (2): La relación entre un vector, el punto de anclaje del vector y el punto que coincide con la ubicación del extremo del vector.

Vector de posición

Existe una representación especial de vectores que utiliza el punto de origen (0,0,0) como punto de anclaje del vector. El vector de posición **v**= <a1,a2,a3> se representa con un segmento de línea entre dos puntos, el origen y B, de modo que:

Punto de origen = (0,0,0)

B = (a1,a2,a3)

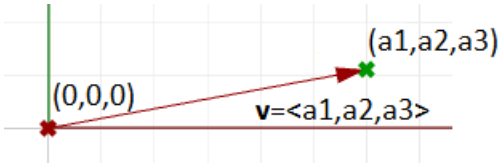


Figura (3): Vector de posición. Las coordenadas del punto extremo son iguales a los componentes del vector correspondiente.

> Un vector de posición de un vector dado ***v***= <a1,a2,a3> es una representación especial del segmento de línea desde el punto de origen (0,0,0) al punto (a1,a2,a3).

Vectores y puntos

Es importante no confundir vectores con puntos. Son conceptos muy diferentes. Los vectores, como mencionamos, representan una cantidad que tiene dirección y longitud, mientras que los puntos indican una ubicación. Por ejemplo, la dirección norte es un vector, mientras que el polo norte es una ubicación (punto).

Si tenemos un vector y un punto que tienen los mismos componentes, como por ejemplo:

v = <3,1,0>

P = (3,1,0)

Podemos dibujar el vector y el punto de la siguiente manera:

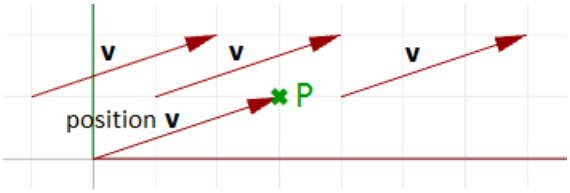


Figura (4): Un vector define una dirección y una longitud. Un punto define una ubicación.

Longitud vectorial

Como se ha mencionado anteriormente, los vectores tienen longitud. Usaremos |**a**| para representar la longitud de un vector dado **a**. Por ejemplo:

a = <4, 3, 0>

$|\mathbf{a}| = \sqrt{(4^2 + 3^2 + 0^2)}$

$|\mathbf{a}| = 5$

En general, la **longitud** de un vector **a**<a1,a2,a3> se calcula de la siguiente manera:

$|\mathbf{a}| = \sqrt{(a1^2 + a2^2 + a3^2)}$

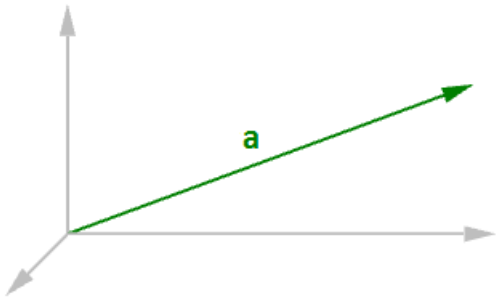


Figura (5): Longitud del vector.

Vector unitario

Un vector unitario es un vector con una longitud igual a una unidad. Los vectores unitarios se utilizan comúnmente para comparar las direcciones de los vectores.

Un vector unitario es un vector cuya longitud es igual a una unidad.

Para calcular un vector unitario, necesitamos encontrar la longitud del vector dado y luego dividir los componentes del vector por la longitud. Por ejemplo:

a = <4, 3, 0>

$|\mathbf{a}| = \sqrt{(4^2 + 3^2 + 0^2)}$

$|\mathbf{a}| = 5$ unit length

Si **b** = vector unitario de **a**, entonces:

b = <4/5, 3/5, 0/5>

b = <0.8, 0.6, 0>

$|\mathbf{b}| = \sqrt{(0.8^2 + 0.6^2 + 0^2)}$

$|\mathbf{b}| = \sqrt{(0.64 + 0.36 + 0)}$

$|\mathbf{b}| = \sqrt{(1)} = 1$ unit length

En general:

a = <a1, a2, a3>

El vector unitario de **a** = <a1/|**a**|, a2/|**a**|, a3/|**a**|>

Figura (6): El vector unitario es igual a una unidad de longitud del vector.

Operaciones vectoriales

Operación escalar de vectores

La operación escalar de vectores implica multiplicar un vector por un número. Por ejemplo:

a = <4, 3, 0>

2***a*** = <2*4, 2*3, 2*0>

2***a*** = <8, 6, 0>

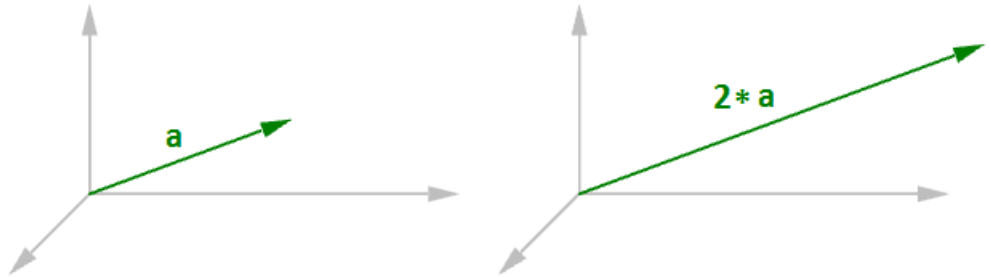


Figura (7): Operación escalar de vectores

En general, dado el vector **a**= <a1, a2, a3> y un número real *t*

*t****a*** = < *t*a1, *t*a2, *t*a3 >

Suma de vectores

La suma de vectores toma dos vectores y se obtiene un tercer vector. Sumamos vectores añadiendo sus componentes.

> Los vectores se suman añadiendo sus componentes.

Por ejemplo, si tenemos dos vectores:

a<1, 2, 0>

b<4, 1, 3>

a+**b** = <1+4, 2+1, 0+3>

a+**b** = <5, 3, 3>

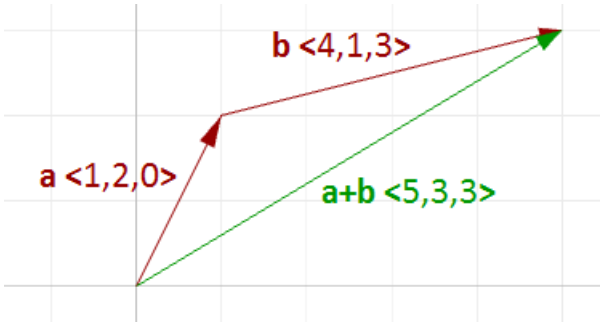


Figura (8): Suma de vectores.

En general, la suma de vectores de los dos vectores **a** y **b** se calcula de la siguiente manera:

a = <a1, a2, a3>

b = <b1, b2, b3>

a+**b** = <a1+b1, a2+b2, a3+b3>

La suma de vectores es útil para hallar la dirección media de dos o más vectores. En estos casos, solemos usar vectores de la misma longitud. Este es un ejemplo que muestra la diferencia entre emplear vectores de iguales o de diferentes longitudes en la resultante de la suma de dichos vectores:

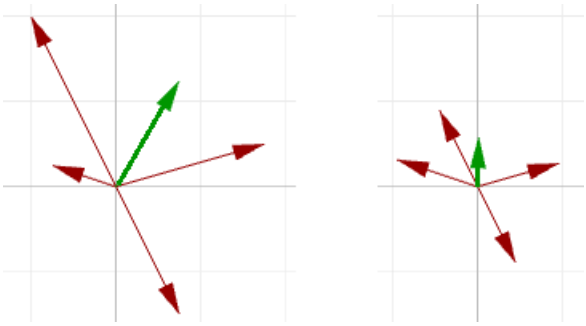


Figura (9): Suma de varios vectores de longitud (izquierda). Suma de vectores de la misma longitud (derecha) para obtener la dirección media.

Es probable que los vectores de entrada no tengan la misma longitud. Para encontrar la dirección media, tiene que utilizar el vector unitario de los vectores de entrada. Como se ha mencionado anteriormente, el vector unitario es un vector de que tiene una longitud igual a 1.

Resta de vectores

La resta de vectores toma dos vectores y se obtiene un tercer vector. Restamos dos vectores restando los componentes correspondientes. Por ejemplo, si tenemos dos vectores **a** y **b** y restamos **b** de **a**, entonces:

a<1, 2, 0>

b<4, 1, 4>

a-**b** = <1-4, 2-1, 0-4>

a-**b** = <-3, 1, -4>

Si restamos **b** de **a**, obtenemos un resultado diferente:

b - **a** = <4-1, 1-2, 4-0>

b - **a** = <3, -1, 4>

Tenga en cuenta que el vector **b - a** tiene la misma longitud que el vector **a - b**, pero va en la dirección opuesta.

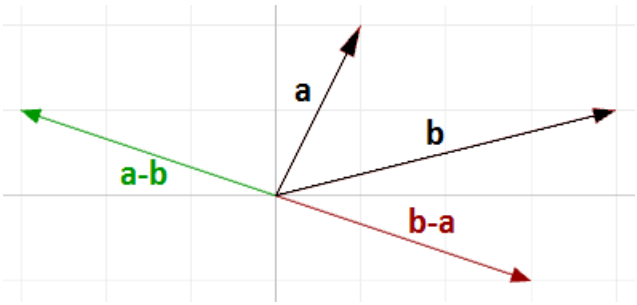


Figura (10): Resta de vectores.

En general, si tenemos dos vectores, **a** y **b**, entonces **a** - **b** es un vector que se calcula de la siguiente manera:

a = <a1, a2, a3>

b = <b1, b2, b3>

a - **b** = <a1 - b1, a2 - b2, a3 - b3>

La resta de vectores se suele usar para hallar vectores entre puntos. Por lo tanto, si necesitamos hallar un vector que vaya desde el punto extremo del vector de posición **b** al punto extremo del vector de posición **a**, usamos la resta de vectores (**a-b**) como se muestra en la Figura (11).

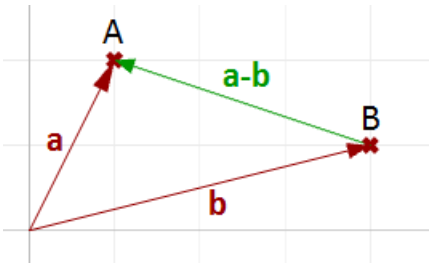


Figura (11): Utilice la resta de vectores para hallar un vector entre dos puntos.

Propiedades de los vectores

Los vectores tienen ocho propiedades fundamentales. Si **a**, **b** y **c** son vectores, y *s* y *t* son números, entonces:

a + **b** = **b** + **a**

a + 0 = **a**

s * (**a** + **b**) = *s* * **a** + *s* * **b**

s * *t* * (**a**) = *s* * (*t* * **a**)

a + (**b** + **c**) = (**a** + **b**) + **c**

a + (-**a**) = 0

(*s* + *t*) * **a** = *s* * **a** + *t* * **a**

1 * **a** = **a**

Producto escalar de vectores

El producto escalar toma dos vectores y se obtiene un número.

Por ejemplo, si tenemos los dos vectores **a** y **b** de modo que:

a = <1, 2, 3>

b = <5, 6, 7>

Entonces el producto escalar es la suma de multiplicar los componentes de la siguiente manera:

a · **b** = 1 * 5 + 2 * 6 + 3 * 7

a · **b** = 38

En general, dados los dos vectores **a** y **b**:

a = <a1, a2, a3>

b = <b1, b2, b3>

a · **b** = a1 * b1 + a2 * b2 + a3 * b3

Siempre obtenemos un número positivo del producto escalar entre dos vectores cuando van en la misma dirección general. Un producto escalar negativo entre dos vectores significa que los dos vectores van en la dirección general opuesta.

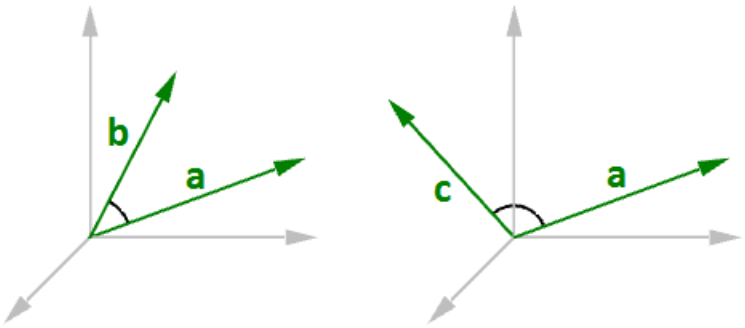


Figura (12): Cuando los dos vectores van en la misma dirección (izquierda), el resultado es un producto escalar positivo. Cuando los dos vectores van en la dirección opuesta (derecha), el resultado es un producto escalar negativo.

Al calcular el producto escalar de dos vectores unitarios, el resultado siempre está entre -1 y 1. Por ejemplo:

a = <1, 0, 0>

b = <0.6, 0.8, 0>

a · **b** = (1 * 0.6, 0 * 0.8, 0 * 0) = 0.6

Además, el producto escalar de un vector consigo mismo es igual a la longitud de ese vector elevado a dos. Por ejemplo:

a = <0, 3, 4>

a · **a** = 0 * 0 + 3 * 3 + 4 * 4

a · **a** = 25

Cálculo de la longitud cuadrada del vector **a**:

$|\mathbf{a}| = \sqrt{(4^2 + 3^2 + 0^2)}$

$|\mathbf{a}| = 5$

$|\mathbf{a}|^2 = 25$

Producto escalar de vectores, longitudes y ángulos

Existe una relación entre el producto escalar de dos vectores y el ángulo entre ellos.

> El producto escalar de dos vectores unitarios válidos (distintos a cero) es igual al coseno del ángulo entre ellos.

En general:

$\mathbf{a} \cdot \mathbf{b} = |\mathbf{a}| * |\mathbf{b}| * \cos(\theta)$, or

$\mathbf{a} \cdot \mathbf{b} / (|\mathbf{a}| * |\mathbf{b}|) = \cos(\theta)$

Donde:

θ es el ángulo incluido entre los vectores.

Si los vectores**a** y **b** son vectores unitarios, podemos decir que:

$\mathbf{a} \cdot \mathbf{b} = \cos(\theta)$

Y dado que el coseno de un ángulo de 90 grados es igual a 0, podemos decir que:

> Los vectores *a* y *b* son ortogonales si, y solo si, $a \cdot b = 0$.

Por ejemplo, si calculamos el producto escalar de los dos vectores ortogonales, el eje x y el eje y universales, el resultado será igual a cero.

$\mathbf{x} = <1, 0, 0>$

$\mathbf{y} = <0, 1, 0>$

$\mathbf{x} \cdot \mathbf{y} = (1 * 0) + (0 * 1) + (0 * 0)$

$\mathbf{x} \cdot \mathbf{y} = 0$

También existe una relación entre el producto escalar y la longitud de proyección de un vector sobre otro. Por ejemplo:

$\mathbf{a} = <5, 2, 0>$

$\mathbf{b} = <9, 0, 0>$

unidad(**b**) = <1, 0, 0>

$\mathbf{a} \cdot$ unit(**b**) = (5 * 1) + (2 * 0) + (0 * 0)

$\mathbf{a} \cdot$ unidad(**b**) =5 (que es igual a la longitud de proyección de **a** sobre **B**)

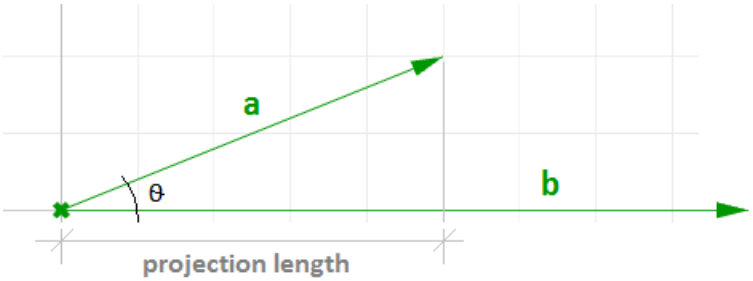


Figura (13): El producto escalar es igual a la longitud de proyección de un vector sobre un vector unitario distinto a cero.

En general, dado un vector **a** y un vector distinto a cero **b**, podemos calcular la longitud de proyección *pL* del vector **a** sobre el vector **b** utilizando el producto escalar.

$pL = |\mathbf{a}| * \cos(\theta)$

$pL = \mathbf{a} \cdot \text{unidad}(\mathbf{b})$

Propiedades del producto escalar

S i**a**, **b** y **c** son vectores y *s* es un número, entonces:

$\mathbf{a} \cdot \mathbf{a} = |\mathbf{a}|^2$

$\mathbf{a} \cdot (\mathbf{b} + \mathbf{c}) = \mathbf{a} \cdot \mathbf{b} + \mathbf{a} \cdot \mathbf{c}$

$0 \cdot \mathbf{a} = 0$

$\mathbf{a} \cdot \mathbf{b} = \mathbf{b} \cdot \mathbf{a}$

$(s * \mathbf{a}) \cdot \mathbf{b} = s * (\mathbf{a} \cdot \mathbf{b}) = \mathbf{a} \cdot (s * \mathbf{b})$

Producto vectorial

El producto vectorial toma dos vectores y se obtiene un tercer vector que es ortogonal a ambos.

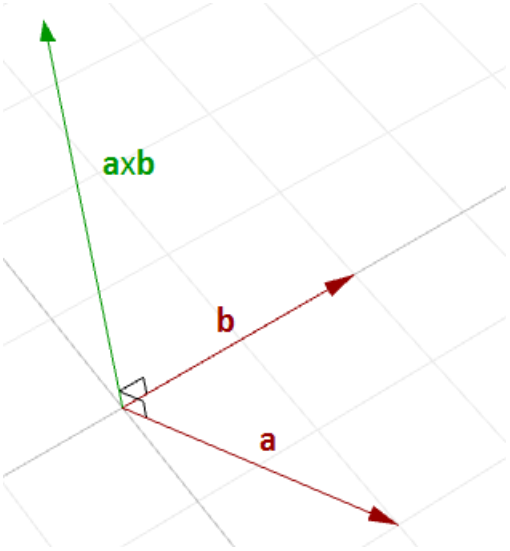


Figura (14): Cálculo del producto vectorial de dos vectores.

Por ejemplo, si tiene dos vectores que se encuentran en el plano xy universal, entonces su producto vectorial es un vector perpendicular al plano xy que va en la dirección del eje z universal positivo o negativo. Por ejemplo:

$\mathbf{a} = <3, 1, 0>$

$\mathbf{b} = <1, 2, 0>$

$\mathbf{a} \times \mathbf{b} = < (1 * 0 - 0 * 2), (0 * 1 - 3 * 0), (3 * 2 - 1 * 1) >$

$\mathbf{a} \times \mathbf{b} = <0, 0, 5>$

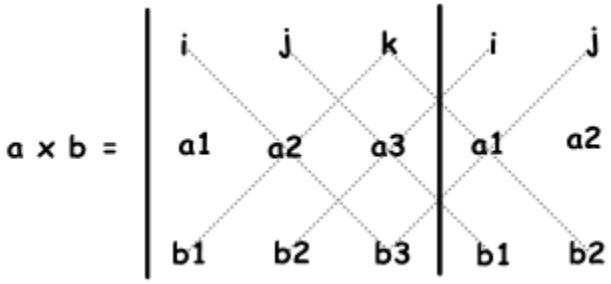
El vector *a* x *b* es ortogonal a ambos *a* y *b.*

Probablemente nunca tenga que calcular manualmente un producto vectorial de dos vectores, pero si tiene curiosidad sobre cómo se hace, continúe leyendo; si no, puede omitir esta sección. El producto vectorial $\mathbf{a} \times \mathbf{b}$ se resuelve utilizando *determinantes*. Esta ilustración simple muestra cómo calcular un determinante utilizando los vectores base estándar:

$\mathbf{i} = <1, 0, 0>$

j = <0,1, 0>

k = <0, 0, 1>



El producto vectorial de los dos vectores **a**<a1, a2, a3> y **b**<b1, b2, b3> se calcula de la siguiente manera utilizando el diagrama anterior:

a × **b** = **i** (a2 * b3) + **j** (a3 * b1) + **k** (a1 * b2) - **k** (a2 * b1) - **i** (a3 * b2) - **j** (a1 * b3)

a × **b** = **i** (a2 * b3 - a3 * b2) + **j** (a3 * b1 - a1 * b3) + **k** (a1 * b2 - a2 * b1)

a × **b** = <a2 * b3 – a3 * b2, a3 * b1 - a1 * b3, a1 * b2 - a2 * b1 >

Producto vectorial y ángulo entre vectores

Existe una relación entre el ángulo entre dos vectores y la longitud de su producto vectorial. Cuanto menor sea el ángulo (seno más pequeño), más corto será el producto vectorial. El orden de los operandos es importante en los productos vectoriales. Por ejemplo:

a = <1, 0, 0>

b = <0, 1, 0>

a × **b** = <0, 0, 1>

b × **a** = <0, 0, -1>

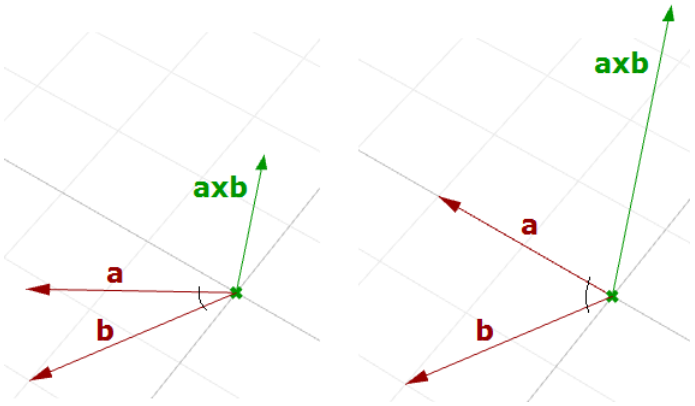
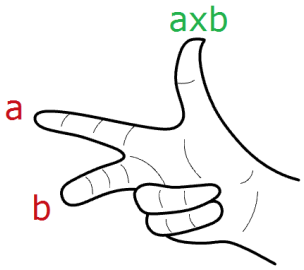


Figura (15): Relación entre el seno del ángulo entre dos vectores y la longitud de su vector de producto vectorial.

En el sistema para diestros de Rhino, la dirección de $\mathbf{a} \times \mathbf{b}$ viene dada por la regla de la mano derecha (donde **a**= dedo índice, **b**= dedo corazón y $\mathbf{a} \times \mathbf{b}$ = pulgar).



En general, para cualquier par de vectores 3D **a** y **b**:

$$|\mathbf{a} \times \mathbf{b}| = |\mathbf{a}|\,|\mathbf{b}| \sin(\theta)$$

Ubicación:

θ es el ángulo incluido entre los vectores de posición de **a** y **b**

Si **a** y **b** son vectores unitarios, entonces podemos decir que la longitud de su producto vectorial es igual al seno del ángulo entre ellos. En otras palabras:

$$|\mathbf{a} \times \mathbf{b}| = \sin(\theta)$$

El producto vectorial entre dos vectores nos ayuda a determinar si dos vectores son paralelos. Esto se debe a que el resultado es siempre un vector cero.

Los vectores *a* y *b* son paralelos si, y solo si, $a \times b = 0$.

Propiedades del producto vectorial

Si **a**, **b** y **c** son vectores, y *s* es un número, entonces:

$\mathbf{a} \times \mathbf{b} = -\mathbf{b} \times \mathbf{a}$

$(s * \mathbf{a}) \times \mathbf{b} = s * (\mathbf{a} \times \mathbf{b}) = \mathbf{a} \times (s * \mathbf{b})$

$\mathbf{a} \times (\mathbf{b} + \mathbf{c}) = \mathbf{a} \times \mathbf{b} + \mathbf{a} \times \mathbf{c}$

$(\mathbf{a} + \mathbf{b}) \times \mathbf{c} = \mathbf{a} \times \mathbf{c} + \mathbf{b} \times \mathbf{c}$

$\mathbf{a} \cdot (\mathbf{b} \times \mathbf{c}) = (\mathbf{a} \times \mathbf{b}) \cdot \mathbf{c}$

$\mathbf{a} \times (\mathbf{b} \times \mathbf{c}) = (\mathbf{a} \cdot \mathbf{c}) * \mathbf{b} - (\mathbf{a} \cdot \mathbf{b}) * \mathbf{c}$

Ecuación vectorial de una línea

La ecuación vectorial de una línea se utiliza en aplicaciones de modelado 3D y diseño gráfico por ordenador.

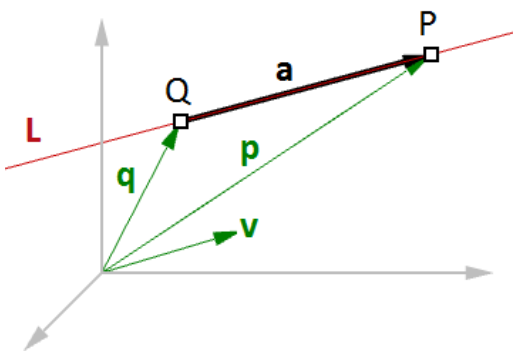


Figura (16): Ecuación vectorial de una línea.

Por ejemplo, si sabemos la dirección de una línea y un punto en esa línea, podemos hallar cualquier otro punto en la línea utilizando vectores, como se muestra a continuación:

L = línea
v = <a, b, c> vector unitario de dirección de la línea
Q = (x0, y0, z0) punto de posición de la línea
P = (x, y, z) cualquier punto en la línea

Sabemos que:

a = *t* * **v** --- (2)

p = **q** + **a** --- (1)

A partir de 1 y 2:

p = **q** + *t* * **v** --- (3)

Sin embargo, también podemos expresar (3) de la siguiente manera:

<x, y, z> = <x0, y0, z0> + <*t* * a, *t* * b, *t* * c>

<x, y, z> = <x0 + *t* * a, y0 + *t* * b, z0 + *t* * c>

Por tanto:

x = x0 + *t* * a
y = y0 + *t* * b
z = z0 + *t* * c

Que es lo mismo que:

P = Q + *t* * **v**

> Dado un punto Q y una dirección v en una línea, cualquier punto P de esa línea se puede calcular utilizando la ecuación vectorial de una línea P = Q + t * v donde *t* es un número.

Otro ejemplo común es hallar el punto medio entre dos puntos. A continuación, se muestra cómo encontrar el punto medio utilizando la ecuación vectorial de una línea:

q es el vector de posición del punto Q

p es el vector de posición del punto P

a es el vector que va de Q a P

De la resta de vectores se obtiene que:

a = **p** - **q**

De la ecuación lineal se obtiene que:

M = Q + *t* * **a**

Y como necesitamos hallar el punto medio, entonces:

t = 0.5

Por tanto, podemos decir que:

M = Q + 0.5 * **a**

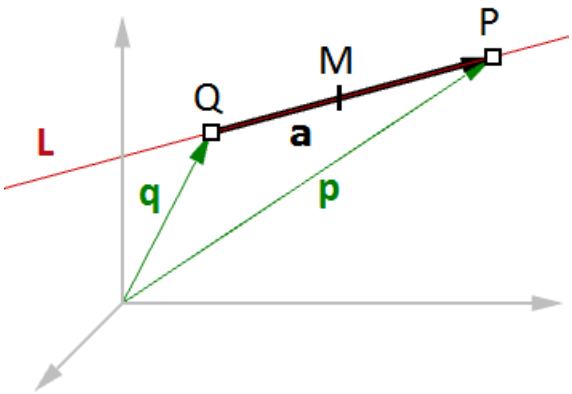


Figura (17): Búsqueda del punto medio entre dos puntos de entrada.

En general, puede hallar cualquier punto entre Q y P cambiando el valor *t* entre 0 y 1 utilizando la ecuación general:

M = Q + *t* * (P - Q)

> Dados dos puntos Q y P, cualquier punto M entre los dos puntos se calcula utilizando la ecuación M = Q + *t* * (P - Q) donde *t* es un número entre 0 y 1.

Ecuación vectorial de un plano

Una forma de definir un plano es cuando hay un punto y un vector que es perpendicular al plano. Ese vector generalmente se conoce como *normal* al plano. La normal apunta en la dirección por encima del plano.

Un ejemplo de cómo calcular la normal de un plano es cuando hay tres puntos no lineales en el plano.

En la Figura (16), dados:

A = el primer punto del plano

B = el segundo punto del plano

C = el tercer punto del plano

Y:

a = un vector de posición del punto A

B = un vector de posición del punto B

C = un vector de posición del punto C

Podemos encontrar el vector de la normal **n** de la siguiente manera:

n = (**b** - **a**) × (**c** - **a**)

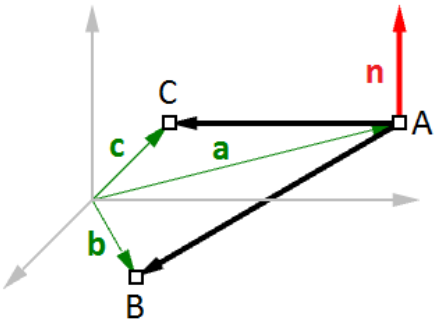


Figura (18): Vectores y planos

También podemos derivar la ecuación escalar del plano utilizando el producto escalar del vector:

$\mathbf{n} \cdot (\mathbf{b} - \mathbf{a}) = 0$

Si:

$\mathbf{n} = \langle a, b, c \rangle$

$\mathbf{b} = \langle x, y, z \rangle$

$\mathbf{a} = \langle x0, y0, z0 \rangle$

Entonces podemos ampliar lo anterior:

$\langle a, b, c \rangle \cdot \langle x-x0, y-y0, z-z0 \rangle = 0$

Resolver el producto escalar da la ecuación escalar general de un plano:

$a * (x - x0) + b * (y - y0) + c * (z - z0) = 0$

Tutoriales

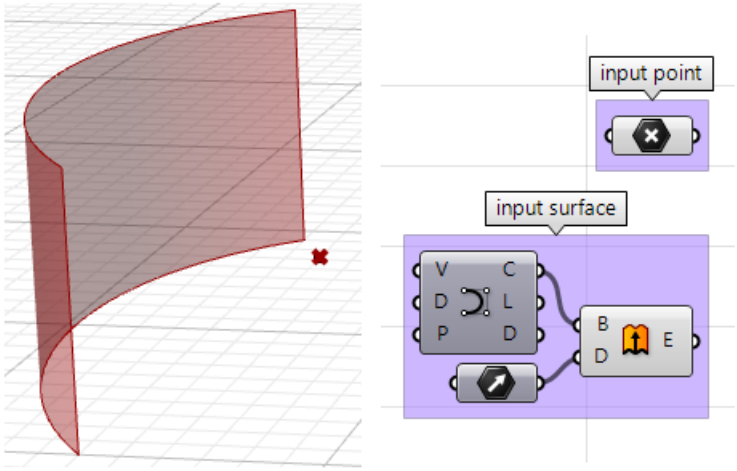
Todos los conceptos tratados en este capítulo tienen una aplicación directa para resolver problemas geométricos comunes que se encuentran durante el modelado. Los siguientes tutoriales paso a paso emplean los conceptos aprendidos en este capítulo utilizando Rhinoceros y Grasshopper (GH).

Dirección de cara

Dados un punto y una superficie, ¿cómo podemos determinar si el punto está orientado hacia la parte frontal o posterior de esa superficie?

Entrada:

1. Una superficie
2. Un punto



Parámetros:

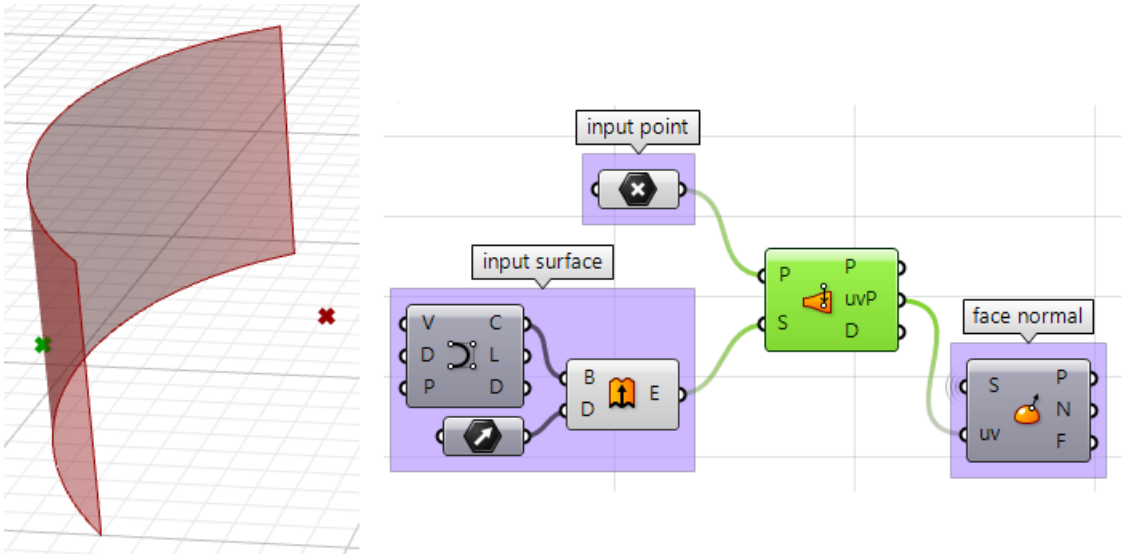
La dirección de la cara está definida por la dirección normal de la superficie. Necesitaremos la siguiente información:

- La dirección normal de la superficie en una ubicación de superficie más cercana al punto de entrada.
- Una dirección de vector desde el punto más cercano al punto de entrada.

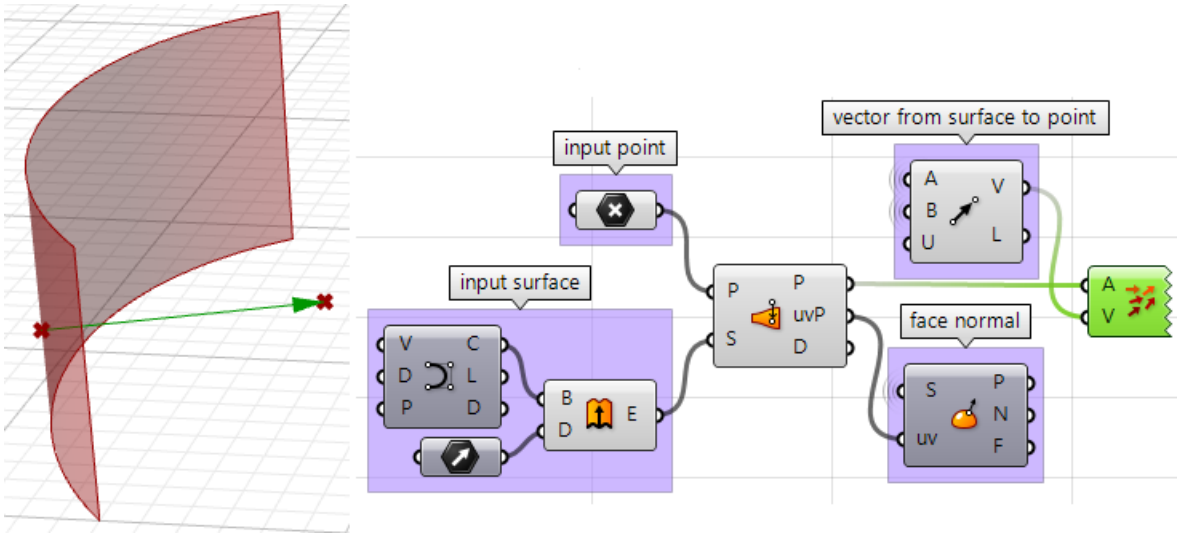
Compare las dos direcciones anteriores. Si van en la misma dirección, el punto está orientado hacia la parte frontal; de lo contrario, está orientado hacia la parte posterior.

Solución:

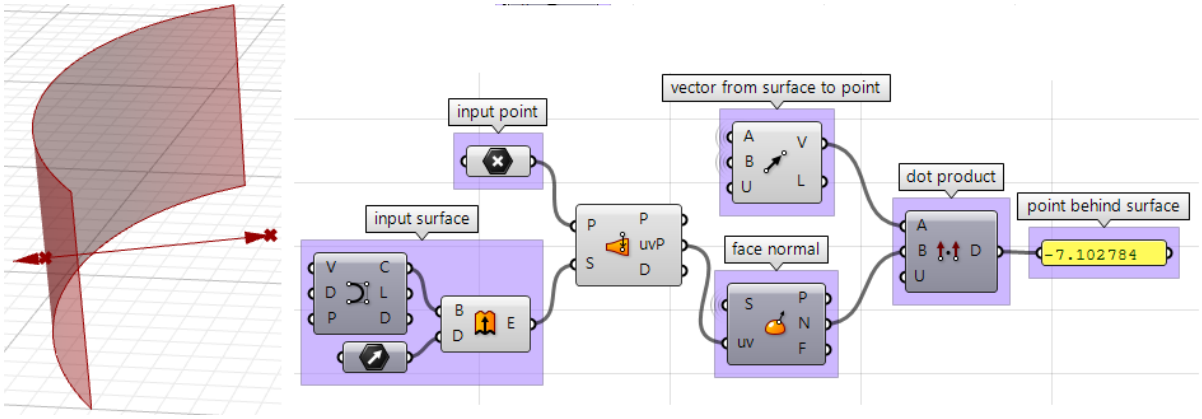
1. Encuentre la ubicación del punto más cercano en la superficie en relación con el punto de entrada utilizando el componente **Pull**. Esto nos dará la ubicación uv del punto más cercano, que luego podemos usar para evaluar la superficie y hallar su dirección normal.



2. Ahora podemos usar el punto más cercano para dibujar un vector orientado hacia el punto de entrada. También podemos dibujar:

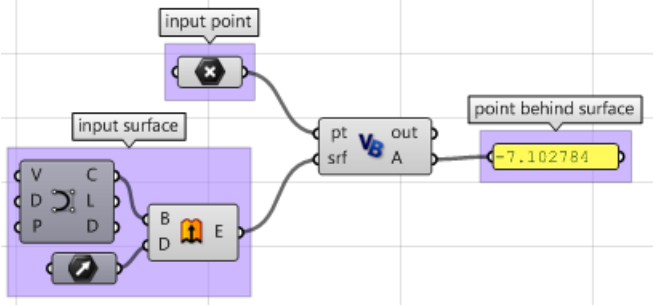


3. Podemos comparar los dos vectores utilizando el producto escalar. Si el resultado es positivo, el punto está delante de la superficie. Si el resultado es negativo, el punto está detrás de la superficie.



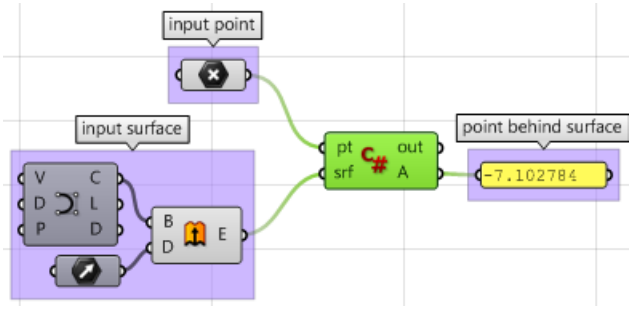
Los pasos anteriores también se pueden resolver utilizando otros lenguajes de scripting.

Uso del componente VB de Grasshopper:



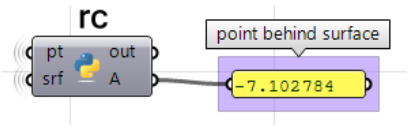
```
Private Sub RunScript(ByVal pt As Point3d, ByVal srf As Surface, ByRef A As Object)

  'Decalre variables
  Dim u, v As Double
  Dim closest_pt As Point3d

  'get closest point u, v
  srf.ClosestPoint(pt, u, v)

  'get closest point
  closest_pt = srf.PointAt(u, v)

  'calculate direction from closest point to test point
  Dim dir As New Vector3d(pt - closest_pt)

  'calculate surface normal
  Dim normal = srf.NormalAt(u, v)

  'compare the two directions using the dot product
  A = dir * normal

End Sub
```

Uso del C componente C# de Grasshopper:



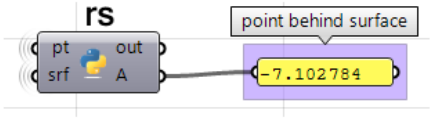

```
private void RunScript(Point3d pt, Surface srf, ref object A)
{
  //Decalre variables
  double u, v;
  Point3d closest_pt;

  //get closest point u, v
  srf.ClosestPoint(pt, out u, out v);

  //get closest point
  closest_pt = srf.PointAt(u, v);

  //calculate direction from closest point to test point
  Vector3d dir = pt - closest_pt;

  //calculate surface normal
  Vector3d normal = srf.NormalAt(u, v);

  //compare the two directions using the dot product
  A = dir * normal;
}
```

Uso del componente Grasshopper Python y el SDK de RhinoCommon:




```
#find the closest point
found, u, v = srf.ClosestPoint(pt)

if found:

    #get closest point
    closest_pt = srf.PointAt(u, v)

    #calculate direction from closest point to test point
    dir = pt - closest_pt

    #calculate surface normal
    normal = srf.NormalAt(u, v)

    #compare the two directions using the dot product
    A = dir * normal
```

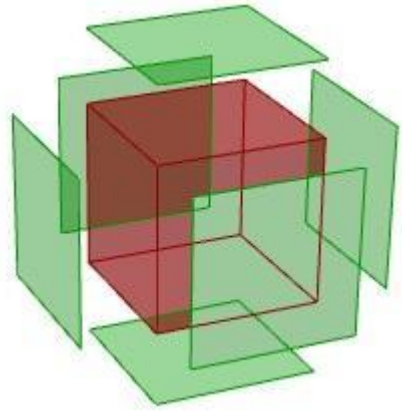
Uso del componente Grasshopper Python y la librería RhinoScriptSyntax:




```
#import RhinoScript library
import rhinoscriptsyntax as rs

#find the closest point
u, v = rs.SurfaceClosestPoint(srf, pt)

#get closest point
closest_pt = rs.EvaluateSurface(srf, u, v)

#calculate direction from closest point to test point
dir = rs.PointCoordinates(pt) - closest_pt

#calculate surface normal
normal = rs.SurfaceNormal(srf, [u, v])

#compare the two directions using the dot product
A = dir * normal
```

Caja descompuesta

El siguiente tutorial muestra cómo descomponer una polisuperficie. Así es como queda la caja descompuesta al final:



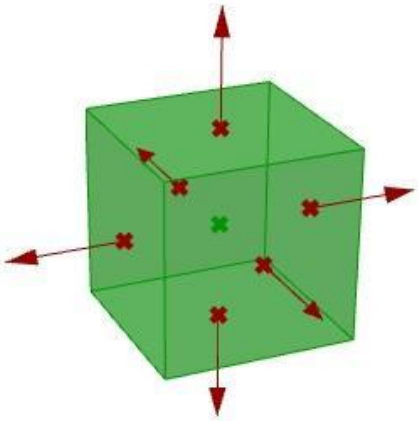
Entrada:

Identifique la entrada, que es una caja. Utilizaremos el parámetro **Box** en GH:

Parámetros:

Piense en todos los parámetros que tenemos que conocer para este tutorial.

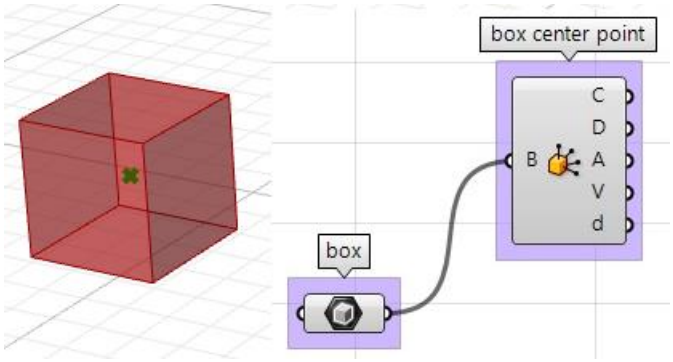
- El centro de la descomposición.
- Las caras de la caja que estamos descomponiendo.
- La dirección hacia la que se mueve cada cara.



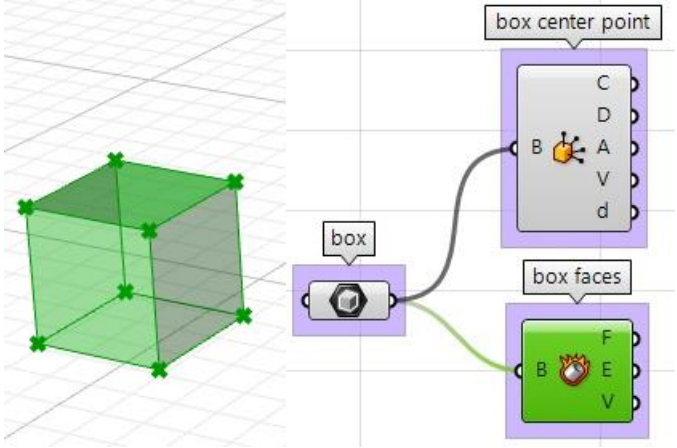
Una vez identificados los parámetros, es cuestión de juntarlos en una solución uniendo los pasos lógicos para lograr una respuesta.

Solución:

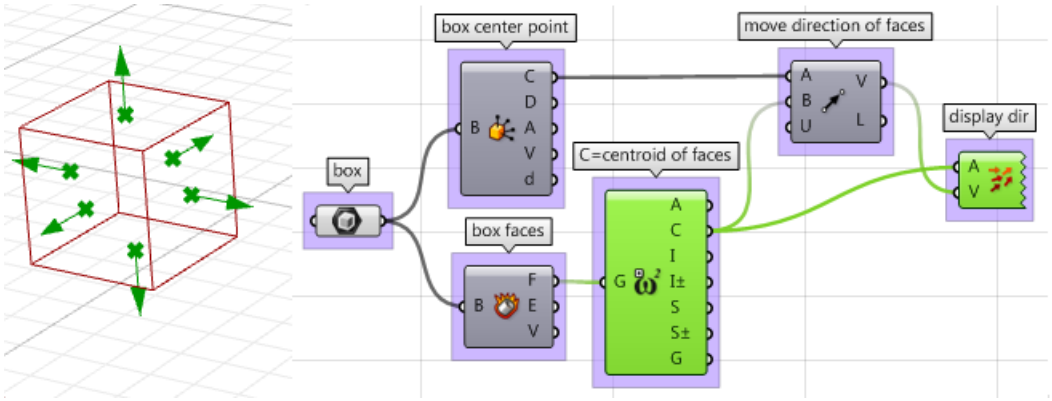
1. Encuentre el centro de la caja utilizando el componente **Box Properties** en GH:



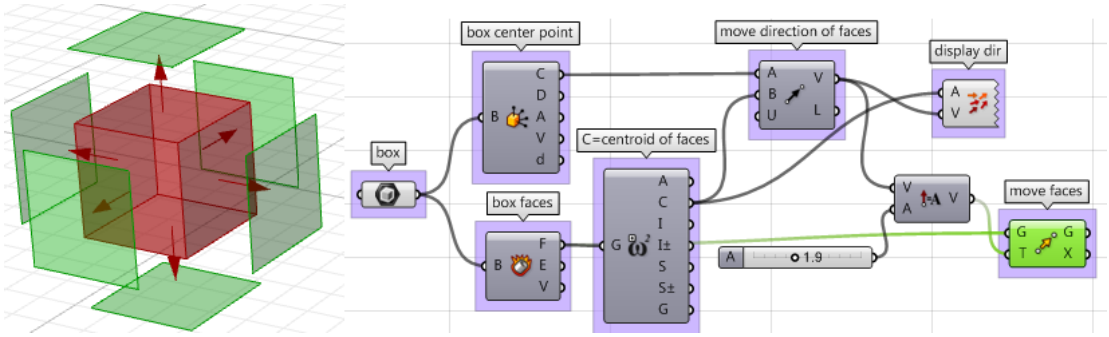
2. Extraiga las caras de la caja con el componente **Deconstruct Brep**:



3. La dirección hacia la que movemos las caras es la parte complicada. Primero debemos encontrar el centro de cada cara y luego definir la dirección desde el centro de la caja hacia el centro de cada cara de la siguiente manera:

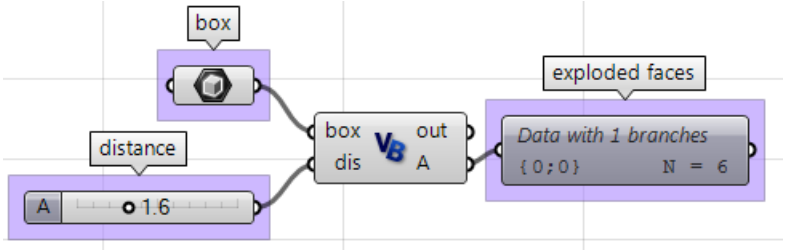


4. Una vez tengamos todos los parámetros listos, podemos usar el componente **Move** para mover las caras en la dirección adecuada. Asegúrese de configurar la amplitud deseada de los vectores y ya podrá empezar.



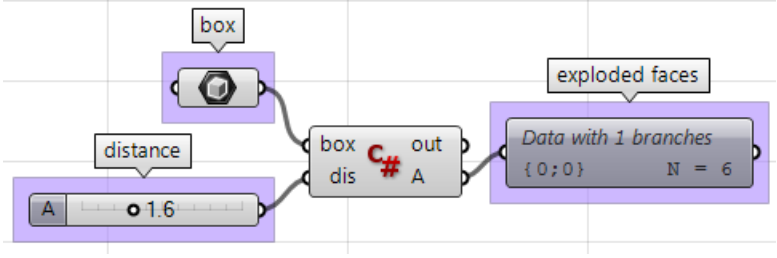
Los pasos anteriores también se pueden resolver mediante VB script, C# o Python. A continuación se muestra la solución con estos lenguajes de scripting.

Uso del componente VB de Grasshopper:



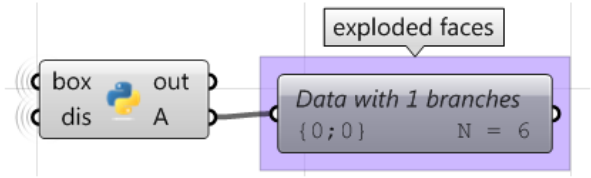
```
Private Sub RunScript(ByVal box As Brep, ByVal dis As Double, ByRef A As Object)

  'get the brep center
  Dim area As Rhino.Geometry.AreaMassProperties
  area = Rhino.Geometry.AreaMassProperties.Compute(box)

  Dim box_center As Point3d
  box_center = area.Centroid

  'get a list of faces
  Dim faces As Rhino.Geometry.Collections.BrepFaceList = box.Faces

  'decalre variables
  Dim center As Point3d
  Dim dir As Vector3d
  Dim exploded_faces As New List( Of Rhino.Geometry.Brep )
  Dim i As Int32
  'loop through all faces

  For i = 0 To faces.Count() - 1
    'extract each of the face
    Dim extracted_face As Rhino.Geometry.Brep = box.Faces.ExtractFace(i)

    'get the center of each face
    area = Rhino.Geometry.AreaMassProperties.Compute(extracted_face)
    center = area.Centroid

    'calculate move direction (from box centroid to face center)
    dir = center - box_center
    dir.Unitize()
    dir *= dis

    'move the extracted face
    extracted_face.Transform(Transform.Translation(dir))

    'add to exploded_faces list
    exploded_faces.Add(extracted_face)
  Next

  'assign exploded list of faces to output
  A = exploded_faces
End Sub
```

Uso del componente C# de Grasshopper:




```
private void RunScript(Brep box, double dis, ref object A)
{

  //get the brep center
  Rhino.Geometry.AreaMassProperties area = Rhino.Geometry.AreaMassProperties.Compute(box);
  Point3d box_center = area.Centroid;

  //get a list of faces
  Rhino.Geometry.Collections.BrepFaceList faces = box.Faces;

  //decalre variables
  Point3d center;
  Vector3d dir;
  List<Rhino.Geometry.Brep> exploded_faces = new List<Rhino.Geometry.Brep>();

  //loop through all faces
  for( int i = 0; i < faces.Count(); i++ )
  {
    //extract each of the face
    Rhino.Geometry.Brep extracted_face = box.Faces.ExtractFace(i);

    //get the center of each face
    area = Rhino.Geometry.AreaMassProperties.Compute(extracted_face);
    center = area.Centroid;

    //calculate move direction (from box centroid to face center)
    dir = center - box_center;
    dir.Unitize();
    dir *= dis;

    //move the extracted face
    extracted_face.Transform(Transform.Translation(dir));

    //add to exploded_faces list
    exploded_faces.Add(extracted_face);
  }

  //assign exploded list of faces to output
  A = exploded_faces;
}
```

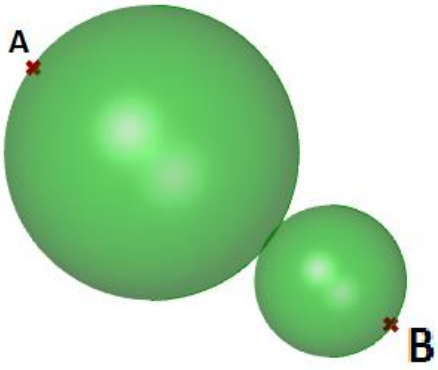
Uso del componente Python de Grasshopper:




```
import Rhino

#get the brep center
area = Rhino.Geometry.AreaMassProperties.Compute(box)
box_center = area.Centroid

#get a list of faces
faces = box.Faces

#decalre variables
exploded_faces = []

#loop through all faces
for i, face in enumerate(faces):

    #get a duplicate of the face
    extracted_face = faces.ExtractFace(i)

    #get the center of each face
    area = Rhino.Geometry.AreaMassProperties.Compute(extracted_face)
    center = area.Centroid

    #calculate move direction (from box centroid to face center)
    dir = center - box_center
    dir.Unitize()
    dir *= dis

    #move the extracted face
    move = Rhino.Geometry.Transform.Translation(dir)
    extracted_face.Transform(move)

    #add to exploded_faces list
    exploded_faces.append(extracted_face)

#assign exploded list of faces to output
A = exploded_faces
```

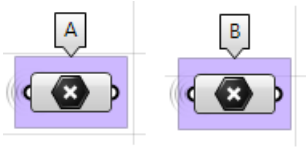
Esferas tangentes

Este tutorial muestra cómo crear dos esferas tangentes entre dos puntos de entrada. El resultado será el siguiente:



Entrada:

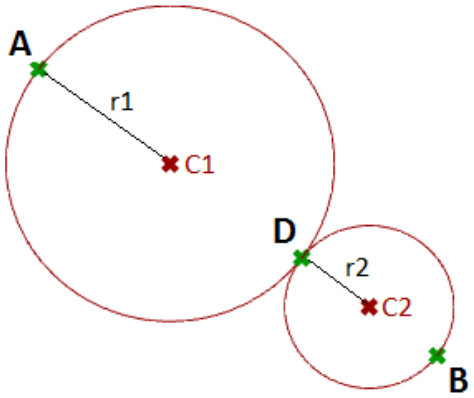
Dos puntos (A y B) en el sistema de coordenadas 3D.



Parámetros:

A continuación, se muestra un diagrama de los parámetros que necesitaremos para resolver el problema:

- Un punto tangente D entre las dos esferas, en algún parámetro *t* (0-1) entre los puntos A y B.
- El centro de la primera esfera o el punto medio C1 entre A y D.
- El centro de la segunda esfera o el punto medio C2 entre D y B.
- El radio de la primera esfera (r1) o la distancia entre A y C1.
- El radio de la segunda esfera (r2) o la distancia entre D y C2.



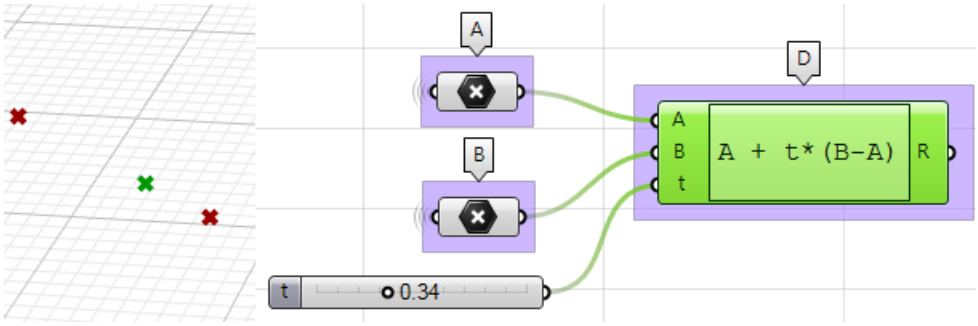
Solución:

1. Utilice el componente **Expression** para definir el punto **D** entre **A** y **B** en algún parámetro **t**. La expresión que usaremos se basa en la ecuación vectorial de una línea: D = A + t*(B-A).

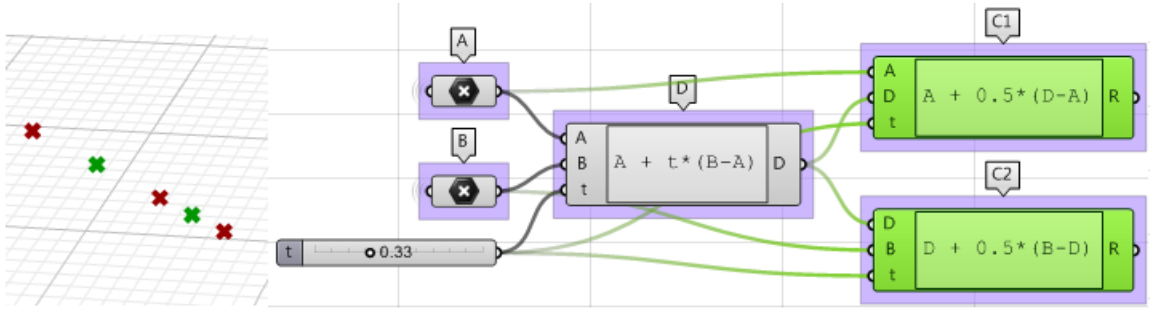
   B-A: es el vector que va de B a A mediante la operación de resta de vectores.

   t*(B-A): donde *t* está entre 0 y 1 para obtener una ubicación en el vector.

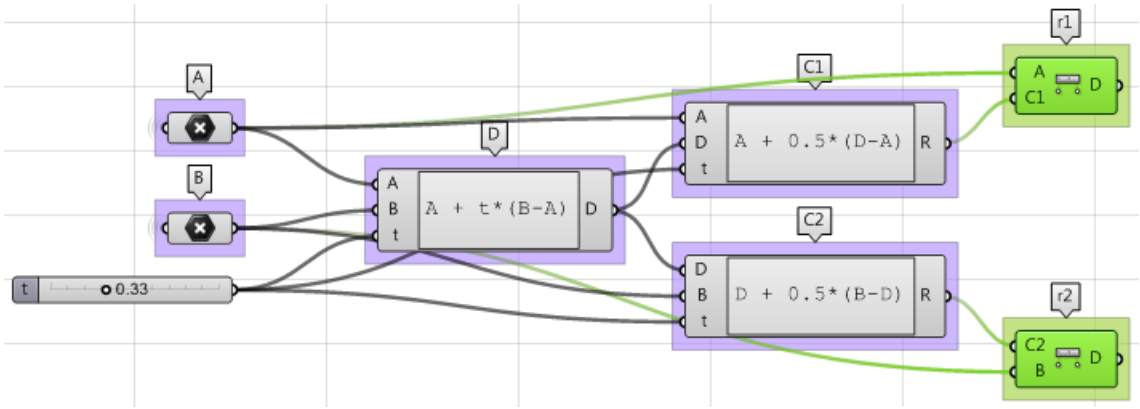
   A+t*(B-A): obtiene un punto en el vector entre A y B.



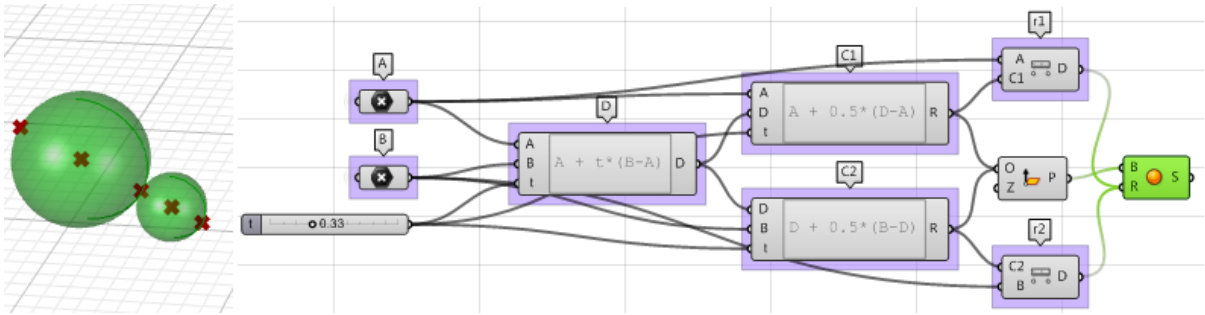
2. Utilice el componente **Expression** para definir también los puntos medios **C1** y **C2**.



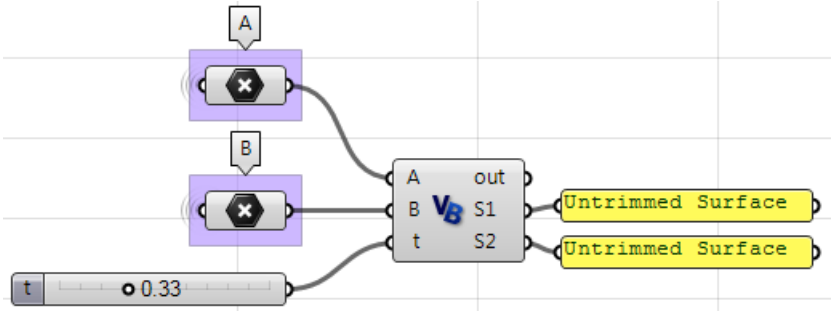
3. El radio de la primera esfera (**r1**) y el radio de la segunda esfera (**r2**) se pueden calcular utilizando el componente **Distance**.



4. El último paso consiste en crear la esfera a partir de un plano base y un radio. Debemos asegurarnos de que los orígenes estén conectados a **C1** y **C2** y al radio de **r1** y **r2**.

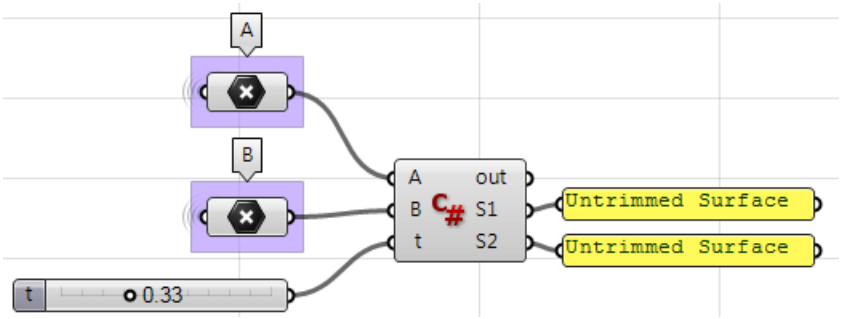


Uso del componente VB de Grasshopper:



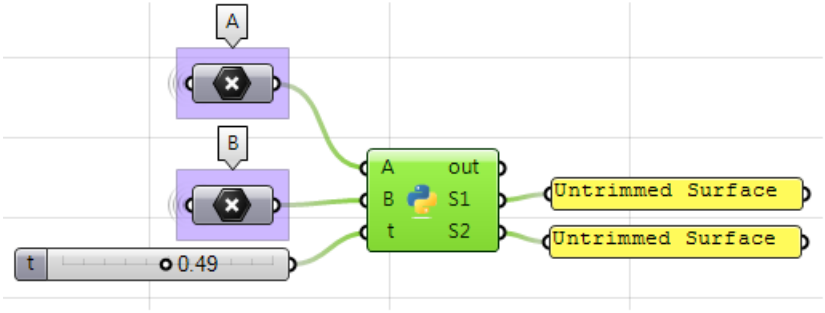

```
Private Sub RunScript(ByVal A As Point3d, ByVal B As Point3d, ByVal t As Double, ByRef S1 As Object, ByRef S2 As Object)

  'declare variables
  Dim D, C1, C2 As Rhino.Geometry.Point3d
  Dim r1, r2 As Double

  'find a point between A and B
  D = A + t * (B - A)

  'find mid point between A and D
  C1 = A + 0.5 * (D - A)

  'find mid point between D and B
  C2 = D + 0.5 * (B - D)

  'find spheres radius
  r1 = A.DistanceTo(C1)
  r2 = B.DistanceTo(C2)

  'create spheres and assign to output
  S1 = New Rhino.Geometry.Sphere(C1, r1)
  S2 = New Rhino.Geometry.Sphere(C2, r2)

End Sub
```

Uso del componente C# de Grasshopper:



```
private void RunScript(Point3d A, Point3d B, double t, ref object S1, ref object S2)
{
  //declare variables
  Rhino.Geometry.Point3d D, C1, C2;
  double r1, r2;

  //find a point between A and B
  D = A + t * (B - A);

  //find mid point between A and D
  C1 = A + 0.5 * (D - A);

  //find mid point between D and B
  C2 = D + 0.5 * (B - D);

  //find spheres radius
  r1 = A.DistanceTo(C1);
  r2 = B.DistanceTo(C2);

  //create spheres and assign to output
  S1 = new Rhino.Geometry.Sphere(C1, r1);
  S2 = new Rhino.Geometry.Sphere(C2, r2);
}
```

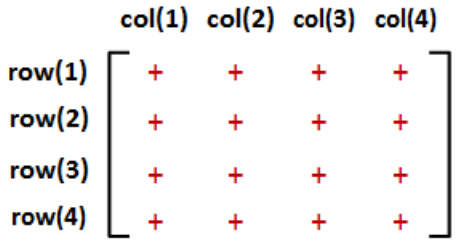
Uso del componente Python de Grasshopper:




```
import Rhino

#find a point between A and B
D = A + t * (B - A)

#find mid point between A and D
C1 = A + 0.5 * (D - A)

#find mid point between D and B
C2 = D + 0.5 * (B - D)

#find spheres radius
r1 = A.DistanceTo(C1)
r2 = B.DistanceTo(C2)

#create spheres and assign to output
S1 = Rhino.Geometry.Sphere(C1, r1)
S2 = Rhino.Geometry.Sphere(C2, r2)
```

2 Matrices y transformaciones

Las *transformaciones* son operaciones como mover (denominado también *traslación*), rotar y escalar objetos. En programación 3D se almacenan utilizando matrices, que son series rectangulares de números. Se pueden realizar múltiples transformaciones rápidamente utilizando matrices. Una matriz [4x4] puede representar todas las transformaciones. Tener una dimensión de matriz unificada para todas las transformaciones ahorra tiempo de cálculo.

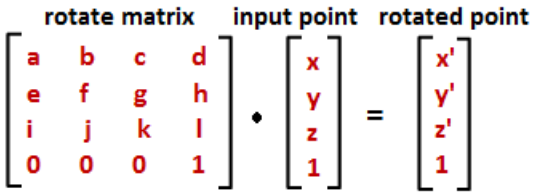


Operaciones con matrices

La operación más relevante en el diseño gráfico por ordenador es la *multiplicación de matrices*. Ahora lo explicamos con más detalle.

Multiplicación de matrices

La multiplicación de matrices se utiliza para aplicar transformaciones a la geometría. Por ejemplo, si tenemos un punto y queremos rotarlo alrededor de un eje, usamos una matriz de rotación y la multiplicamos por el punto para obtener la nueva ubicación rotada.



La mayoría de las veces, necesitamos realizar múltiples transformaciones en la misma geometría. Por ejemplo, si necesitamos mover y rotar mil puntos, podemos usar cualquiera de los siguientes métodos.

Método 1

1. Multiplique la matriz de movimiento por 1000 puntos para mover los puntos.
2. Multiplique la matriz de rotación por los 1000 puntos resultantes para rotar los puntos movidos.

   Número de operaciones =**2000**.

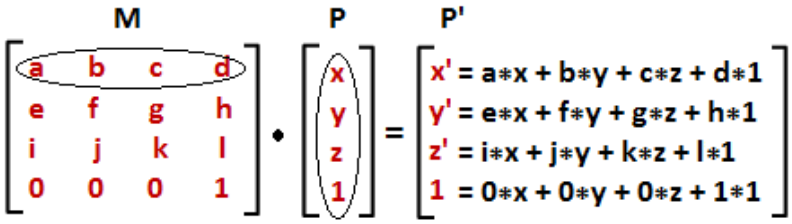
Método 2

1. Multiplique las matrices de rotación y movimiento para crear una matriz de transformación combinada.
2. Multiplique la matriz combinada por 1000 puntos para mover y rotar en un solo paso.

Número de operaciones =**1001**.

Observe que el método 1 requiere casi el doble de operaciones para lograr el mismo resultado. Si bien el método 2 es muy eficaz, solo es posible si las matrices de movimiento y rotación son [4x4]. Por este motivo, en el diseño gráfico por ordenador se usa una matriz [4x4] para representar todas las transformaciones, y una matriz [4x1] para representar puntos.

Las aplicaciones de modelado tridimensional proporcionan herramientas para aplicar transformaciones y multiplicar matrices, pero si tiene curiosidad sobre cómo multiplicar matrices matemáticamente, le explicaremos un ejemplo sencillo. Para multiplicar dos matrices, éstas tienen que coincidir. En otras palabras, el número de columnas de la primera matriz debe ser igual al número de filas de la segunda matriz. La matriz resultante tiene un tamaño igual al número de filas de la primera matriz y al número de columnas de la segunda matriz. Por ejemplo, si tenemos dos matrices, **M** y **P**, con dimensiones iguales a [4x4] y [4x1] respectivamente, la matriz de multiplicación resultante **M · P** tiene una dimensión igual a [4x1] como se muestra en la siguiente ilustración:



Matriz de identidad

La *matriz de identidad* es una matriz especial donde todos los componentes diagonales son iguales a 1 y el resto igual a 0.

1.0	0.0	0.0	0.0
0.0	1.0	0.0	0.0
0.0	0.0	1.0	0.0
0.0	0.0	0.0	1.0

La propiedad principal de la matriz identidad es que si se multiplica por cualquier otra matriz, los valores multiplicados por cero no cambian.

1.0	0.0	0.0	0.0	X	2.0	=	1.0x2.0 + 0.0x3.0 + 0.0x1.0 + 0.0x1.0	=	2.0
0.0	1.0	0.0	0.0		3.0		0.0x2.0 + 1.0x3.0 + 0.0x1.0 + 0.0x1.0		3.0
0.0	0.0	1.0	0.0		1.0		0.0x2.0 + 0.0x3.0 + 1.0x1.0 + 0.0x1.0		1.0
0.0	0.0	0.0	1.0		1.0		0.0x2.0 + 0.0x3.0 + 0.0x1.0 + 1.0x1.0		1.0

Operaciones de transformación

La mayoría de las transformaciones conservan la relación paralela entre las partes de la geometría. Por ejemplo, los puntos colineales permanecen colineales después de la transformación. También los puntos de un plano permanecen coplanares después de la transformación. Este tipo de transformación se denomina *transformacion afin*.

Traslación (mover)

Mover un punto desde una posición inicial a lo largo de un vector determinado se calcula de la siguiente manera:

P' = P + V

Supongamos que:

P (x,y,z) es un punto dado

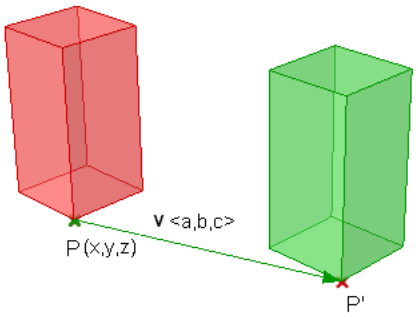
v<a,b,c> es un vector de traslación

Entonces:

P'(x) = x + a

P'(y) = y + b

P'(z) = z + c



Los puntos se representan en un formato de matriz utilizando una matriz [4x1] con un 1 insertado en la última fila. Por ejemplo, el punto P (x,y,z) se representa de la siguiente manera:

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

El uso de una matriz [4x4] para las transformaciones (lo que se denomina un sistema de coordenadas homogéneo) en lugar una matriz [3x3], permite representar todas las transformaciones, incluida la traslación. El formato general de una matriz de traslación es:

$$\begin{bmatrix} 1 & 0 & 0 & a1 \\ 0 & 1 & 0 & a2 \\ 0 & 0 & 1 & a3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Por ejemplo, para mover el punto P (2,3,1) por el vector **v**<2,2,2>, la nueva ubicación del punto es:

P’ = P + **v** = (2+2, 3+2, 1+2) = (4, 5, 3)

Si empleamos las matrices y multiplicamos la matriz de traslación por el punto de entrada, obtenemos la nueva ubicación del punto como se muestra a continuación:

$$\begin{bmatrix} 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 2 \end{bmatrix} \begin{bmatrix} 2 \\ 3 \\ 1 \end{bmatrix} = \begin{bmatrix} (1*2 + 0*3 + 0*1 + 2*1) \\ (0*2 + 1*3 + 0*1 + 2*1) \\ (0*2 + 0*3 + 1*1 + 2*1) \end{bmatrix} = \begin{bmatrix} 4 \\ 5 \\ 3 \end{bmatrix}$$

0 0 0 1 1 (0*2 + 0*3 + 0*1 + 1*1) 1

De manera similar, cualquier geometría se traslada multiplicando sus puntos de construcción por la matriz de traslación. Por ejemplo, si tenemos una caja definida por ocho puntos de esquina y queremos moverla 4 unidades en la dirección x, 5 unidades en la dirección y, y 3 unidades en la dirección z, debemos multiplicar cada uno de los ocho puntos de esquina de la caja por la siguiente matriz de traslación para obtener la nueva caja.

$$\begin{bmatrix} 1 & 0 & 0 & 4 \\ 0 & 1 & 0 & 5 \\ 0 & 0 & 1 & 3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

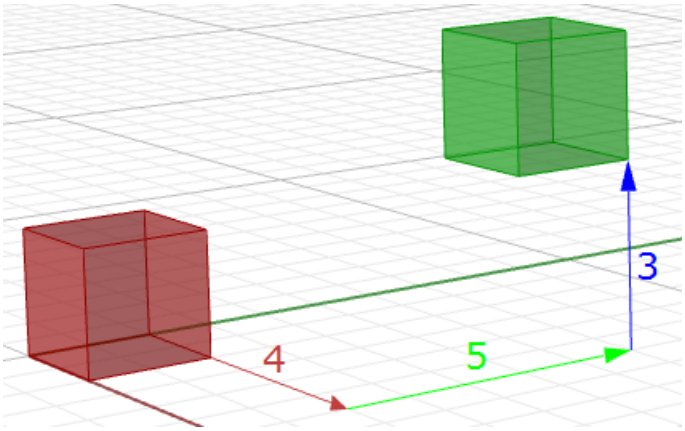


Figura (19): Traslación de todos los puntos de esquina de la caja.

Rotación

Esta sección muestra cómo calcular la rotación alrededor del eje z y el punto de origen utilizando la trigonometría, y luego deducir el formato general de la matriz para la rotación.

Tome un punto en el plano x,y P (x,y) y gírelo un ángulo (b).

Del resultado, podemos deducir lo siguiente:

x = d cos(a) ---(1)

y = d sin(a) ---(2)

x' = d cos(b+a) ---(3)

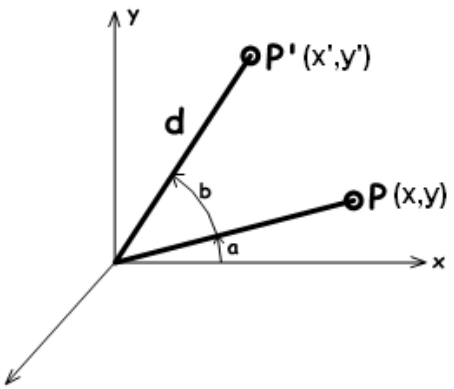
y' = d sin(b+a) --- (4)

Podemos expandir x' e y' usando las identidades trigonométricas del seno y el coseno de la suma de ángulos:

x' = d cos(a)cos(b) - d sin(a)sin(b) ---(5)

y' = d cos(a)sin(b) + d sin(a)cos(b) ---(6)

Usando las ecuaciones 1 y 2:



x' = x cos(b) - y sin(b)
y' = x sin(b) + y cos(b)

La matriz de rotación alrededor del **eje z** quedaría así:

$$\begin{bmatrix} \cos(b) & -\sin(b) & 0 & 0 \\ \sin(b) & \cos(b) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

La matriz de rotación alrededor del **eje x** por el ángulo **b** quedaría así:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(b) & -\sin(b) & 0 \\ 0 & \sin(b) & \cos(b) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

La matriz de rotación alrededor del **eje y** por el ángulo **b** quedaría así:

$$\begin{bmatrix} \cos(b) & 0 & \sin(b) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(b) & 0 & \cos(b) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Por ejemplo, si tenemos una caja y queremos rotarla 30 grados, necesitamos lo siguiente:

1. Construya la matriz de rotación de 30 grados. Utilizando la forma genérica y los valores de coseno y seno del ángulo de 30 grados, la matriz de rotación quedaría así:

$$\begin{bmatrix} 0.87 & -0.5 & 0 & 0 \\ 0.5 & 0.87 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

2. Multiplique la matriz de rotación por la geometría de entrada o, en el caso de una caja, multípliquela por cada uno de los puntos de esquina para hallar la nueva ubicación de la caja.

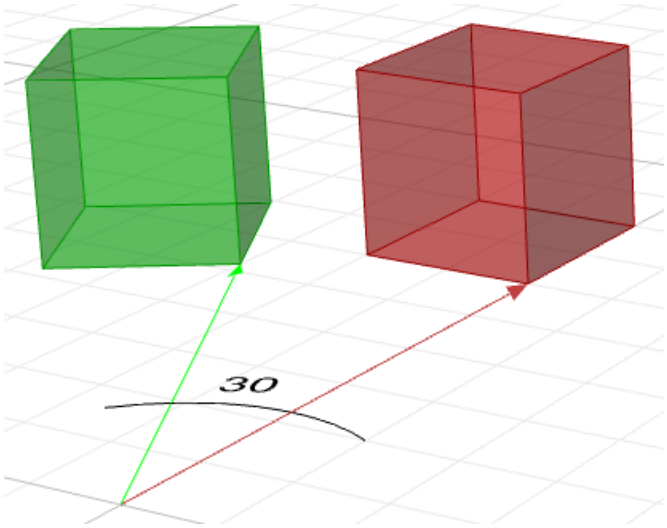


Figura (20): Rotación de geometría.

Escalado

Para escalar la geometría, necesitamos un factor de escala y un centro de escala. El factor de escala puede ser uniforme por igual en las direcciones x,y,z, o puede ser único para cada dimensión. Para escalar un punto, se puede usar la siguiente ecuación:

P' = FactorDeEscalado(S) * P

O bien:

$P'.x = S_x * P.x$

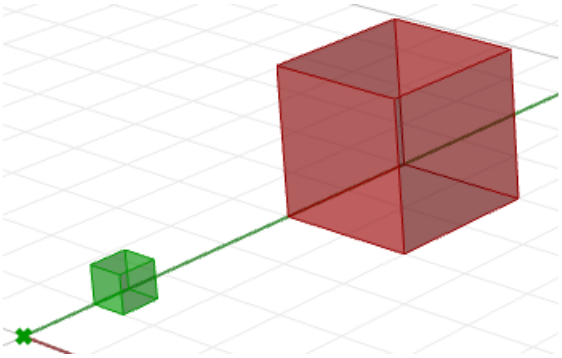
$P'.y = S_y * P.y$

$P'.z = S_z * P.z$

Este es el formato de la matriz de escalado, teniendo en cuenta que el centro del escalado es el punto de origen universal (0,0,0).

$$\begin{bmatrix} Escala\ x & 0 & 0 & 0 \\ 0 & Escala\ y & 0 & 0 \\ 0 & 0 & Escala\ z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Por ejemplo, si quisiéramos escalar una caja 0.25 unidades en relación con el origen universal, la matriz de escalado quedaría así:



0.25	0.00	0.00	0.00
0.00	0.25	0.00	0.00
0.00	0.00	0.25	0.00
0.00	0.00	0.00	1.00

Figura (21): Escalado de geometría

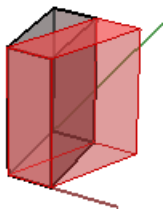
Sesgado

El sesgado en tres dimensiones se mide a lo largo de un par de ejes en relación con el tercero. Por ejemplo, el sesgado en el eje z no va a cambiar la geometría a lo largo de ese eje, sino que altera los valores en x e y. Estos son algunos ejemplos de matrices de sesgado:

1. Sesgado en x y z, manteniendo fija la coordenada y:

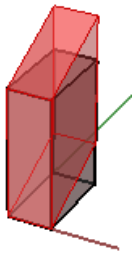
Sesgado en eje x

1.0	0.5	0.0	0.0
0.0	1.0	0.0	0.0
0.0	0.0	1.0	0.0
0.0	0.0	0.0	1.0



Sesgado en eje z

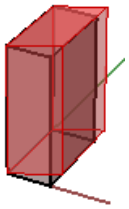
1.0	0.0	0.0	0.0
0.0	1.0	0.0	0.0
0.0	0.5	1.0	0.0
0.0	0.0	0.0	1.0



2. Sesgado en y y z, manteniendo fija la coordenada x:

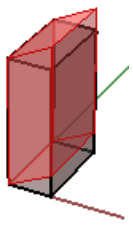
Sesgado en eje y

1.0	0.0	0.0	0.0
0.5	1.0	0.0	0.0
0.0	0.0	1.0	0.0
0.0	0.0	0.0	1.0



Sesgado en eje z

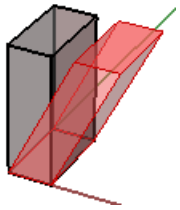
1.0	0.0	0.0	0.0
0.0	1.0	0.0	0.0
0.5	0.0	1.0	0.0
0.0	0.0	0.0	1.0



3. Sesgado en x e y, manteniendo fija la coordenada z:

Sesgado en eje x

1.0	0.0	0.5	0.0
0.0	1.0	0.0	0.0
0.0	0.0	1.0	0.0
0.0	0.0	0.0	1.0



Sesgado en eje y

1.0	0.0	0.0	0.0
0.0	1.0	0.5	0.0
0.0	0.0	1.0	0.0
0.0	0.0	0.0	1.0

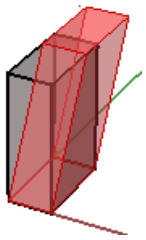
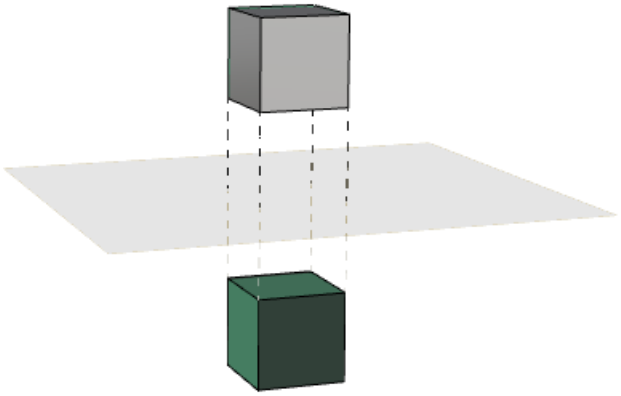


Figura (22): Matrices de sesgado.

Reflejo o espejo

El reflejo o espejo crea un reflejo de un objeto a través de una línea o un plano. Los objetos 2D se reflejan a través de una línea, mientras que los objetos 3D se reflejan en un plano. Tenga en cuenta que la transformación de reflejo cambia la dirección normal de las caras de la geometría.



1.000	0.000	0.000	0.000
0.000	1.000	0.000	0.000
0.000	0.000	-1.000	0.000
0.000	0.000	0.000	1.000

Figura (23): Matriz de reflejo en el plano xy universal. Las direcciones de las caras se invierten.

Proyección plana

Intuitivamente, el punto de proyección de un punto 3D dado P(x,y,z) sobre el plano xy universal es igual a $P_{xy}(x,y,0)$ estableciendo el valor z en cero. De forma similar, la proyección de un punto P en el plano xz es $P_{xz}(x,0,z)$. Al proyectar sobre el plano yz, $P_{xz} = (0,y,z)$. Estas proyecciones se denominan proyecciones ortogonales[1].

Si tenemos una curva como entrada y aplicamos la transformación de proyección plana, obtenemos una curva proyectada sobre ese plano. A continuación se muestra un ejemplo de una curva proyectada sobre el plano xy con el formato de matriz.

Nota: las curvas NURBS (que se explican en el siguiente capítulo) utilizan puntos de control para definir las curvas. La proyección de una curva pasa por la proyección de sus puntos de control.

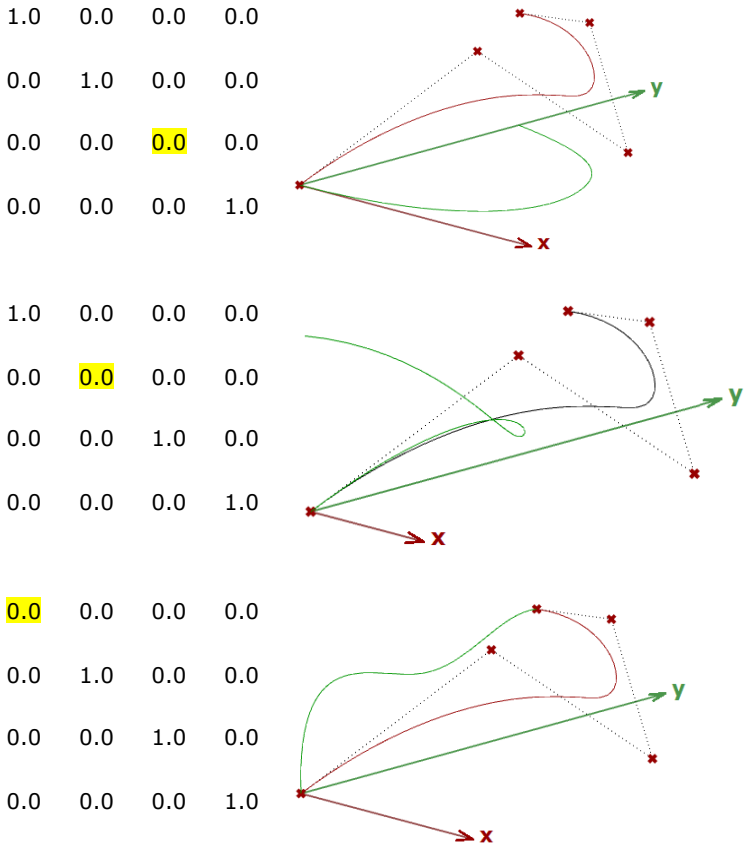


Figura (24): Matrices de proyección.

Tutorial

Transformaciones múltiples

Utilice una matriz para transformar la geometría de la siguiente manera:

[1] Wikipedia: curva Bézier.

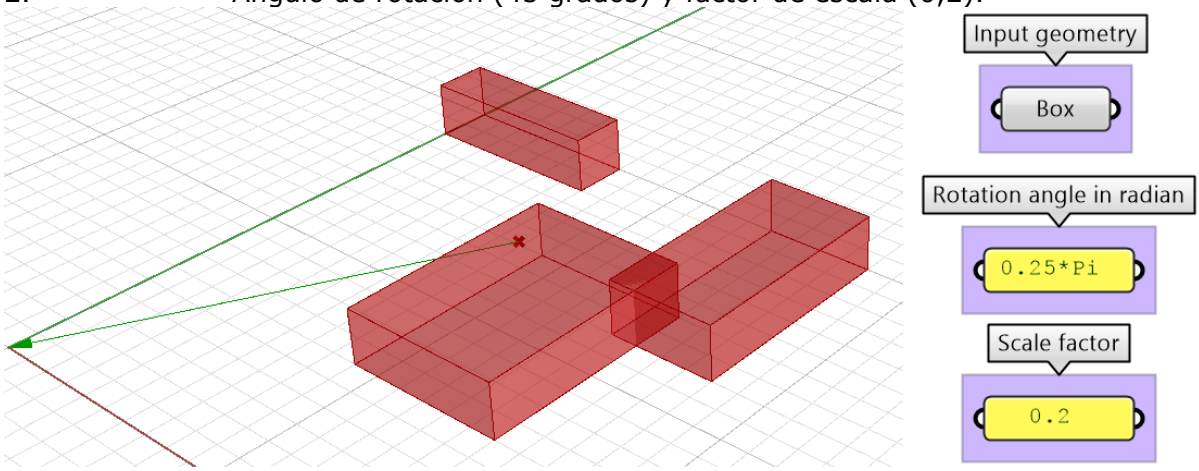
Primero, mueva la geometría de entrada para que su centro esté en el origen, luego gírela 45 grados alrededor del eje z, después escálela uniformemente 0.2 unidades y, finalmente, vuelva a moverla a su ubicación original.

Nota de rendimiento:

Con una gran cantidad de puntos u objetos para transformar, es mucho más eficaz crear una matriz de transformación maestra (multiplicar todas las matrices primero) y después utilizar la matriz maestra resultante una sola vez para transformar todas las entradas, en lugar de transformar varias veces con una matriz cada vez.

Entrada:

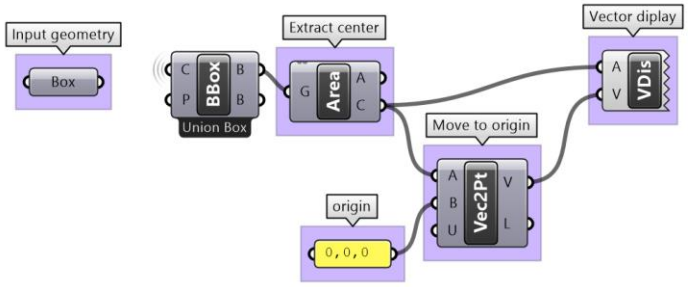
1. Objetos para transformar
2. Ángulo de rotación (45 grados) y factor de escala (0,2).



Entrada adicional:

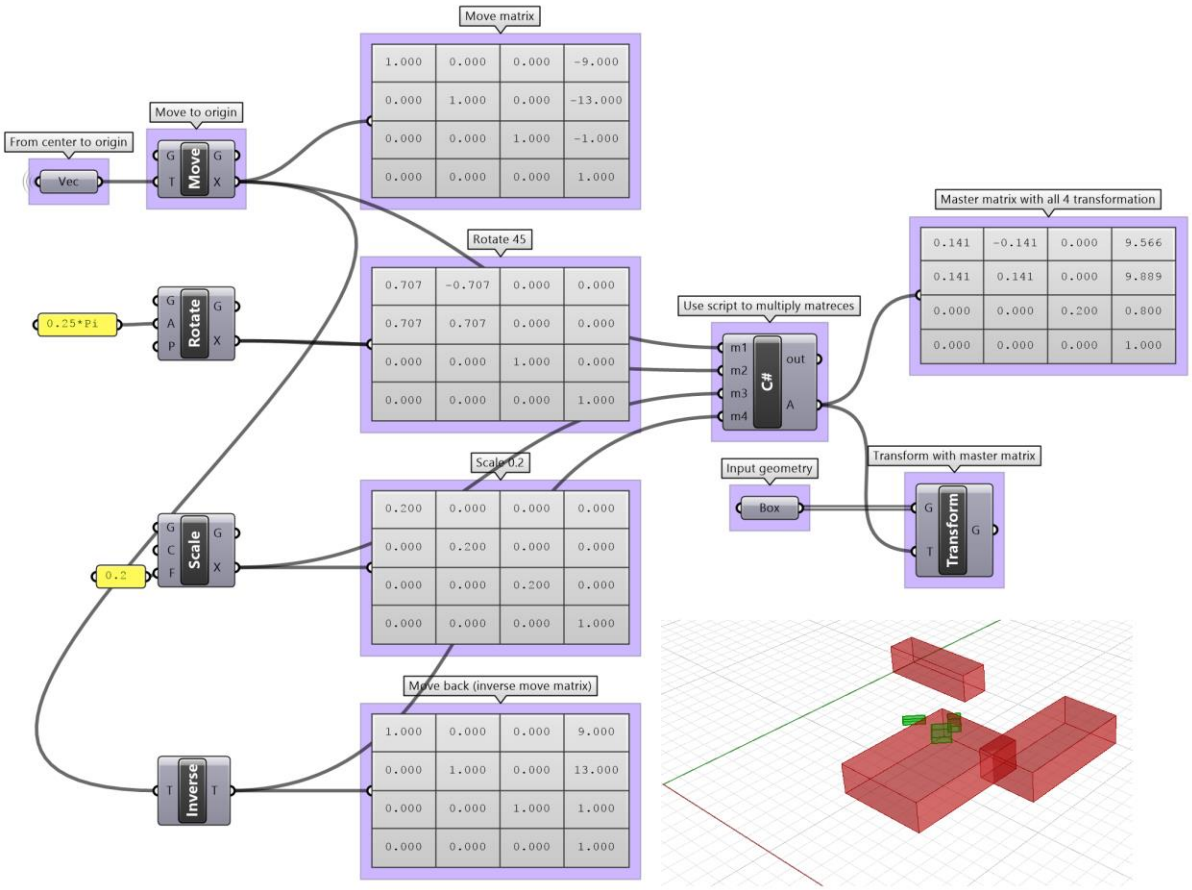
El movimiento inicial necesita lo siguiente:

- Vector desde el centro del cuadro delimitador de entrada hasta el origen.



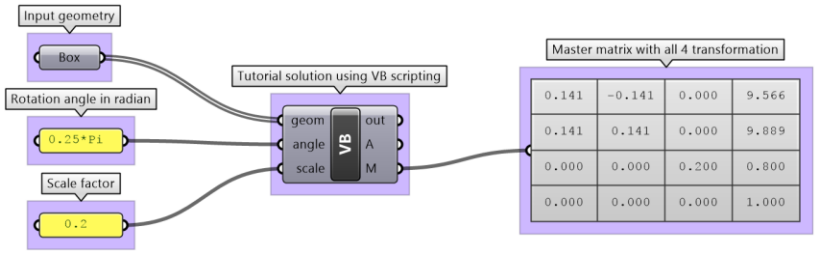
Solución:

1. Genere todas las matrices de transformación: mover, rotar, escalar e inversa del movimiento inicial.
2. Multiplique las matrices desde la última hasta la primera para generar la matriz de transformación maestra.
3. Transforme la entrada utilizando la matriz de transformación maestra.



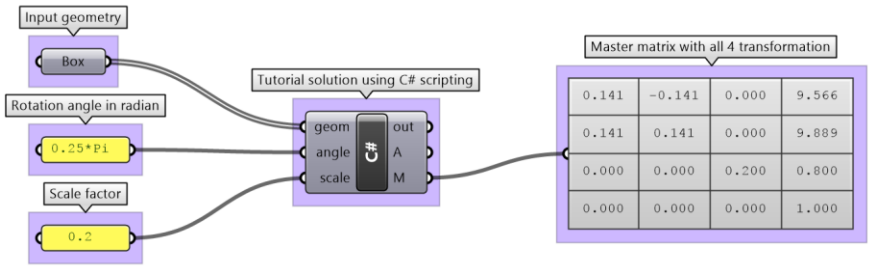
Los pasos anteriores también se pueden resolver mediante scripts.

Uso del componente VB de Grasshopper:



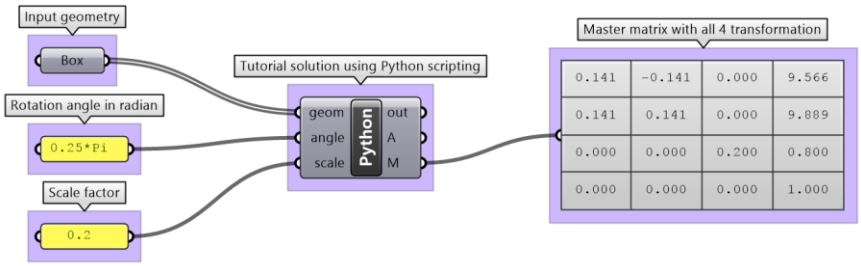

```
Private Sub RunScript(ByVal geom As List(Of GeometryBase), ByVal angle As Double, ByVal scale As Double, ByRef A As Object, ByRef M As Object)

  Dim center As Point3d = Point3d.Unset

  If geom.Count > 0 Then
    Dim bbox As BoundingBox = geom(0).GetBoundingBox(True)

    For i As Integer = 1 To geom.Count - 1
      bbox.Union(geom(i).GetBoundingBox(True))
    Next

    center = bbox.Center
  Else
    Return
  End If

  Dim m1 = Rhino.Geometry.Transform.Translation(Point3d.Origin - center)
  Dim m2 = Rhino.Geometry.Transform.Rotation(angle, Point3d.Origin)
  Dim m3 = Rhino.Geometry.Transform.Scale(Point3d.Origin, scale)
  Dim m4 = Rhino.Geometry.Transform.Identity
  m1.TryGetInverse(m4)
  Dim matrix = m4 * m3 * m2 * m1

  For i As Integer = 0 To geom.Count - 1
    geom(i).Transform(matrix)
  Next

  M = matrix
  A = geom

End Sub
```

Uso del componente C# de Grasshopper:




```
private void RunScript(List<GeometryBase> geom, double angle, double scale, ref object A, ref object M)
{
  //Move to origin, find center
  Point3d center = Point3d.Unset;
  if(geom.Count > 0)
  {
    BoundingBox bbox = geom[0].GetBoundingBox(true);
    for(int i = 1; i < geom.Count ; i++)
    {
      bbox.Union(geom[i].GetBoundingBox(true));
    }
    center = bbox.Center;
  }
  else
    return;

  //Create move transformation matrix
  var m1 = Rhino.Geometry.Transform.Translation(Point3d.Origin - center);
  //Create rotation transformation matrix
  var m2 = Rhino.Geometry.Transform.Rotation(angle, Point3d.Origin);
  //Create scale transformation matrix
  var m3 = Rhino.Geometry.Transform.Scale(Point3d.Origin, scale);
  //Create move back transformation matrix
  var m4 = Rhino.Geometry.Transform.Identity;
  m1.TryGetInverse(out m4);

  //Multiply from last to first transformation
  var matrix = m4 * m3 * m2 * m1;

  //Apply transformation to all input geometry
  for(int i = 0; i < geom.Count ; i++)
  {
    geom[i].Transform(matrix);
  }

  //Assign output
  M = matrix;
  A = geom;
}
```

Uso del componente Python de Grasshopper y el SDK de RhinoCommon:




```python
import Rhino

#Move to origin, find center
center = Rhino.Geometry.Point3d.Unset
if geom.Count > 0:
    bbox = geom[0].GetBoundingBox(True)
    for i in range(1, geom.Count):
        bbox.Union(geom[i].GetBoundingBox(True))
    center = bbox.Center

origin = Rhino.Geometry.Point3d.Origin
#Create move transformation matrix
m1 = Rhino.Geometry.Transform.Translation(origin - center)
#Create rotation transformation matrix
m2 = Rhino.Geometry.Transform.Rotation(angle, origin)
#Create scale transformation matrix
m3 = Rhino.Geometry.Transform.Scale(origin, scale)
#Create move back transformation matrix
m4 = Rhino.Geometry.Transform.Identity
result, m4 = m1.TryGetInverse()

#Multiply from last to first transformation
matrix = m4 * m3 * m2 * m1

#Apply transformation to all input geometry
for g in geom:
    g.Transform(matrix)

#Assign output
M = matrix
A = geom
```

3 Curvas y superficies paramétricas

Imagine que va todos los días de la semana desde su casa hasta su trabajo. Sale a las 8:00 de la mañana y llega a las 9:00. Cada momento entre las 8:00 y las 9:00, se encuentra en un punto del camino. Si traza su ubicación cada minuto durante su viaje, puede definir la trayectoria entre su casa y el trabajo conectando los 60 puntos trazados. Suponiendo que viaja exactamente a la misma velocidad todos los días, a las 8:00 estaría en su casa (ubicación de inicio), a las 9:00 estaría en el trabajo (ubicación final) y a las 8:40 estaría en la ubicación exacta de la trayectoria que corresponde al punto 40 del trazado. ¡Felicidades, acaba de definir su primera curva paramétrica! Ha utilizado el *tiempo* como *parámetro* para definir su trayectoria, y por tanto, se puede decir que la curva de trayectoria es una *curva paramétrica*. El intervalo de tiempo que pasa desde el inicio hasta el final (8 a 9) se denomina *dominio de curva* o *intervalo*.

En general, podemos describir la ubicación x, y y z de una curva paramétrica en términos de un parámetro *t* de la siguiente manera:

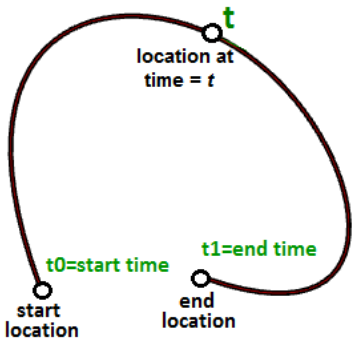
x = x(*t*)

y = y(*t*)

z = z(*t*)

Donde:

t es un intervalo de números reales



Vimos anteriormente que la ecuación paramétrica de una línea en términos del parámetro *t* se define así:

x = x′ + *t* * a
y = y′ + *t* * b
z = z′ + *t* * c

Donde:

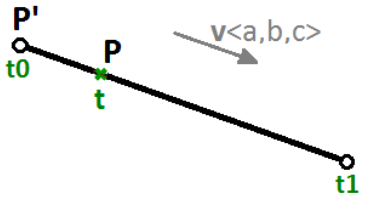
x, y, z son funciones de *t* donde *t* representa un intervalo de números reales.

x ', y' y z' son las coordenadas de un punto en el segmento de línea.

a, b y c definen la pendiente de la línea, de modo que el vector **v**<a, b, c> es paralelo a la línea.

Por lo tanto, podemos escribir la ecuación paramétrica de un segmento de línea usando un parámetro *t* que oscila entre dos valores numéricos reales *t0*,*t1* y un vector unitario **v** que está en la dirección de la línea de la siguiente manera:

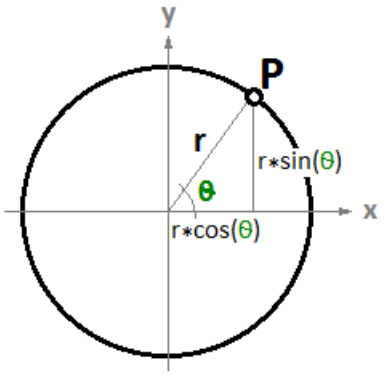
P = P′ + *t* * **v**



Otro ejemplo es un círculo. La ecuación paramétrica del círculo en el plano xy con un centro en el origen (0,0) y un parámetro de ángulo.*t* oscilando entre 0 y 2π radianes es:

$x = r \cos(t)$

$y = r \sin(t)$



Podemos obtener la ecuación general de un círculo desde la paramétrica de la siguiente manera:

$x/r = \cos(t)$

$y/r = \sin(t)$

Y puesto que:

$\cos(t)^2 + \sin(t)^2 = 1$ (identidad de Pitágoras)

Entonces:

$(x/r)^2 + (y/r)^2 = 1$, o

$x^2 + y^2 = r^2$

Curvas paramétricas

Parámetro de curva

Un parámetro de una curva representa la dirección de un punto en esa curva. Como se ha mencionado anteriormente, una curva paramétrica es como una trayectoria recorrida entre dos puntos en una determinada cantidad de tiempo, desplazándose a una velocidad fija o variable. Si el desplazamiento tarda una cantidad de tiempo T, entonces el parámetro *t* representa un tiempo dentro de T que se traduce en una ubicación (punto) en la curva.

Si tiene una trayectoria recta (segmento de línea) entre los dos puntos A y B, y **v** es un vector de A a B (**v** = B - A), entonces puede usar la ecuación de línea paramétrica para hallar todos los puntos M entre A y B de la siguiente manera:

$M = A + t*(B-A)$

Donde:

t es un valor entre 0 y 1.

El intervalo de los valores *t*, de 0 a 1 en este caso, se conoce como *dominio de curva* o *intervalo*. Si *t* es un valor fuera del dominio (menos de 0 o más de 1), entonces el punto resultante M estará fuera de la curva lineal AB.

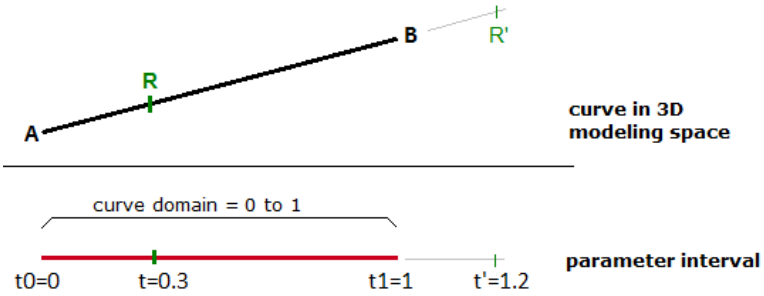


Figura (25): línea paramétrica en el espacio 3D e intervalo de parámetro.

El mismo principio se aplica a cualquier curva paramétrica. Cualquier punto de la curva se puede calcular utilizando el parámetro *t* dentro del intervalo o dominio de valores que definen los límites de la curva. El parámetro de inicio del dominio se suele denominar t0 y el final del dominio t1.

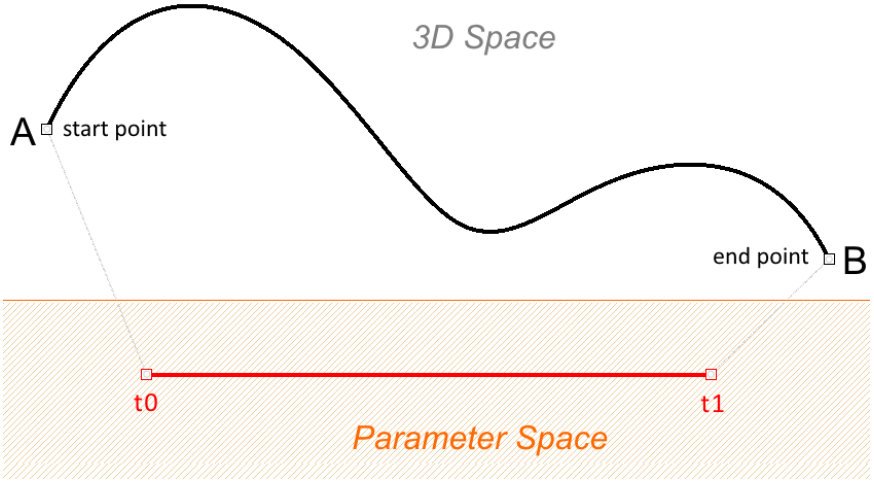


Figura (26): Curva en el espacio 3D y su dominio en el espacio paramétrico.

Dominio de la curva o intervalo

Un *dominio* de curva o *intervalo* es el rango de parámetros que se traduce en un punto dentro de esa curva. El dominio generalmente se describe con dos números reales que definen los límites del dominio expresados en la forma (mínimo a máximo) o (mínimo, máximo). Los límites del dominio pueden ser dos valores cualesquiera que pueden estar relacionados o no con la longitud real de la curva. En un dominio creciente, el parámetro de dominio mínimo calcula el punto inicial de la curva y el dominio máximo calcula el punto final de la curva.

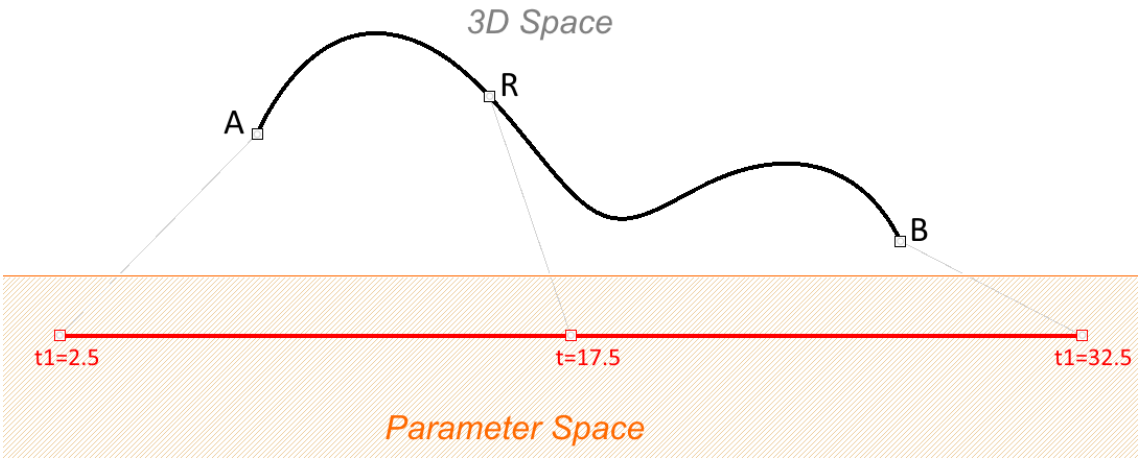


Figura (27): El dominio de curva o intervalo es un conjunto de dos números que generalmente es ascendente. Cuando es posible, la longitud del dominio se establece cerca de la longitud de la curva 3D, pero se puede establecer en cualquier longitud sin cambiar la curva 3D.

El cambio de un dominio de curva se conoce como el proceso de *reparametrizar* la curva. Por ejemplo, es muy común cambiar el dominio de (0 a 1). La reparametrización de una curva no afecta a la forma de la curva 3D. Como cuando se recorre un camino corriendo o caminando, cambia el tiempo que se tarda, pero no cambia la forma del camino.

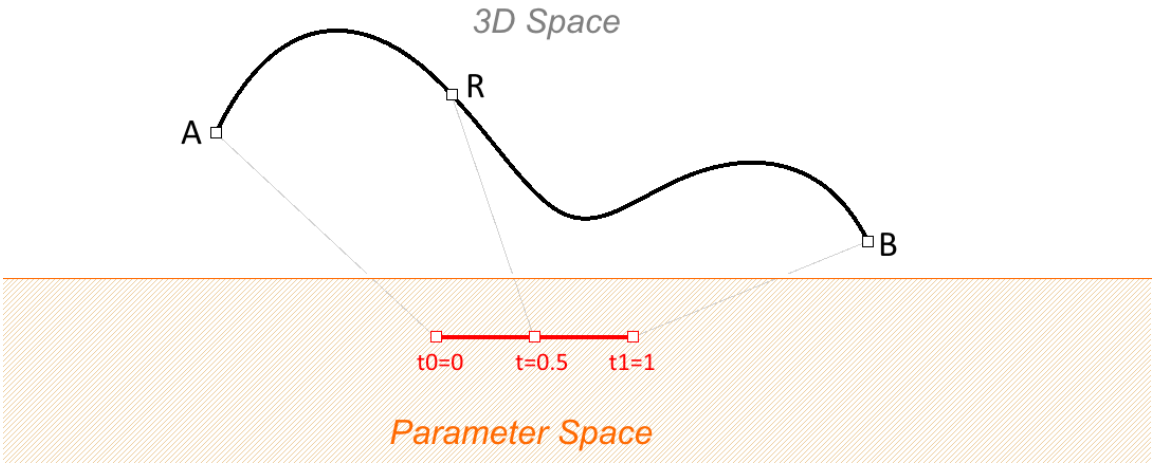


Figura (28): El dominio de curva se puede normalizar (definido en 0 a 1).Tenga en cuenta que si la longitud de la curva 3D es mucho mayor que la longitud del dominio (en un factor de 10 o más), el cálculo de un parámetro podría no producir una ubicación precisa en la curva 3D.

Un dominio creciente significa que el valor mínimo del dominio apunta al inicio de la curva. Los dominios suelen aumentar, pero no siempre.

Cálculo de curvas

Hemos aprendido que un intervalo de curva es el rango de todos los valores de los parámetros suelen ser puntos dentro de la curva 3D. Sin embargo, no hay garantía de que el cálculo en el medio del dominio, por ejemplo, resulte en un punto que se encuentre en el medio de la curva.

La parametrización uniforme de una curva es como recorrer un camino a una velocidad constante. Una línea de grado 1 entre dos puntos es un ejemplo en el que los intervalos o parámetros iguales se traducen en intervalos iguales de longitud de arco en la línea, como en la figura (29). En curvas paramétricas, es raro que los intervalos iguales de parámetros se traduzcan en intervalos iguales en la curva 3D.

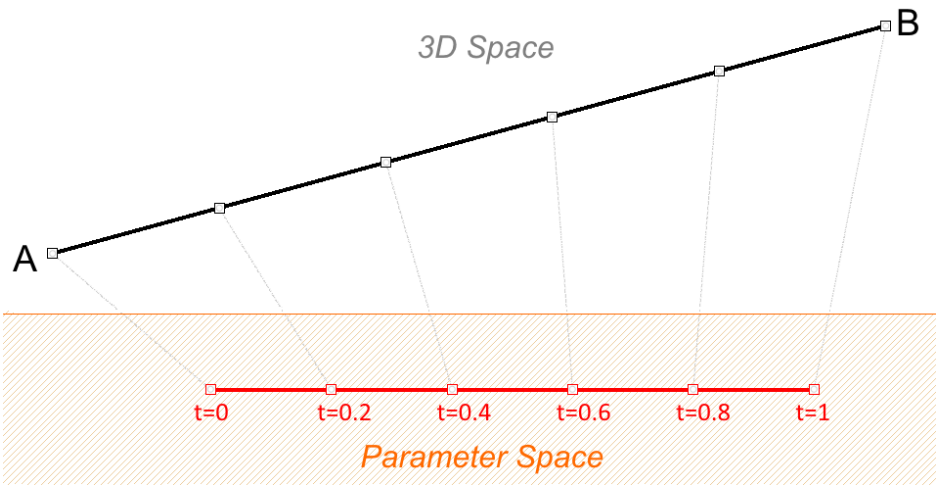


Figura (29): Caso especial de una línea de grado 1 donde los intervalos de parámetros iguales se traducen en longitudes de curva iguales.

Es más probable que la velocidad disminuya o aumente a lo largo del recorrido. Por ejemplo, si tarda 30 minutos en recorrer una carretera, es poco probable que esté exactamente a la mitad en el minuto 15. La figura (30) muestra un caso típico en el que los intervalos de parámetros iguales se traducen en una longitud variable en la curva 3D.

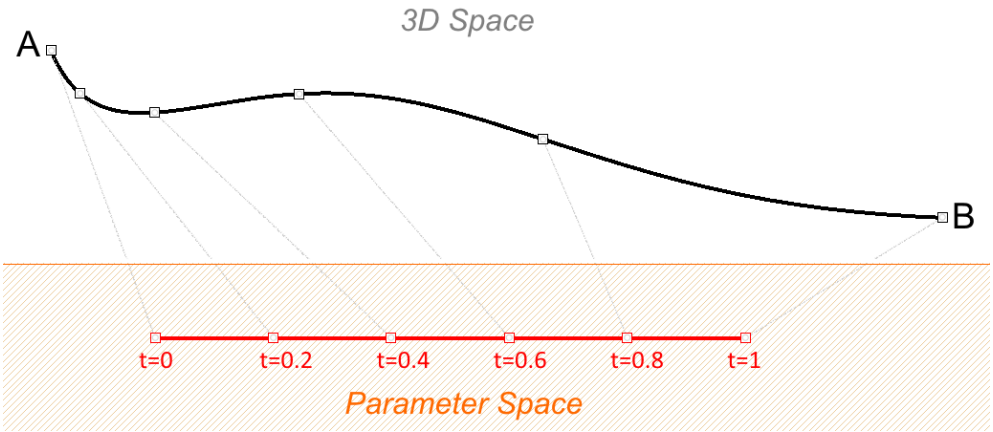


Figura (30): Los intervalos de parámetros iguales no suelen traducirse en distancias iguales en una curva paramétrica, como en las curvas NURBS.

Es posible que tenga que calcular puntos en una curva 3D que se encuentren en un porcentaje definido de la longitud de la curva. Por ejemplo, puede que tenga que dividir la curva en longitudes iguales. Normalmente, los modeladores 3D proporcionan herramientas para calcular curvas en relación con la longitud del arco.

Vector tangente a una curva

Una tangente a una curva en cualquier parámetro (o punto de una curva) es el vector que toca la curva en ese punto, pero no se cruza. La pendiente del vector tangente es igual a la pendiente de la curva en el mismo punto. El siguiente ejemplo calcula la tangente a una curva en dos parámetros diferentes.

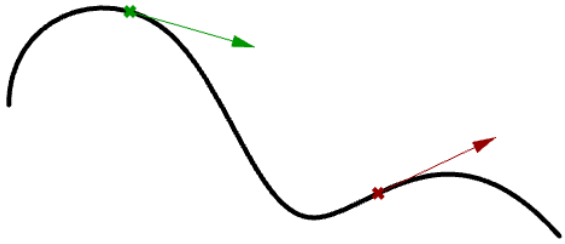


Figura (31): Tangentes a una curva.

Curvas polinomiales cúbicas

Las curvas Hermite[2] y Bézier[3] son dos ejemplos de curvas polinomiales cúbicas determinadas por cuatro parámetros. Una curva Hermite viene determinada por sus dos puntos finales y dos vectores de dirección tangentes a estos puntos, mientras que una curva Bézier está definida por cuatro puntos. A pesar de ser diferentes matemáticamente, comparten características y limitaciones similares.

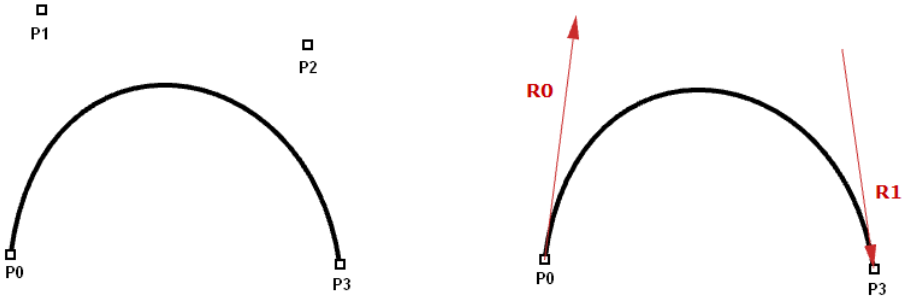


Figura (32): Curvas polinomiales cúbicas. La curva Bézier (izquierda) está definida por cuatro puntos.
La curva Hermite (derecha) está definida por dos puntos y dos tangentes.

[2] Wikipedia: cubic Hermite spline.

[3] Wikipedia: curva Bézier.

En la mayoría de casos, las curvas están compuestas por varios segmentos. Esto requiere lo que se denomina *curva cúbica segmentada*. Aquí se muestra una figura de una curva Bézier que utiliza 7 puntos para crear una curva de tres segmentos. Nótese que, aunque la última curva esta unida, no se ve suave o continua.

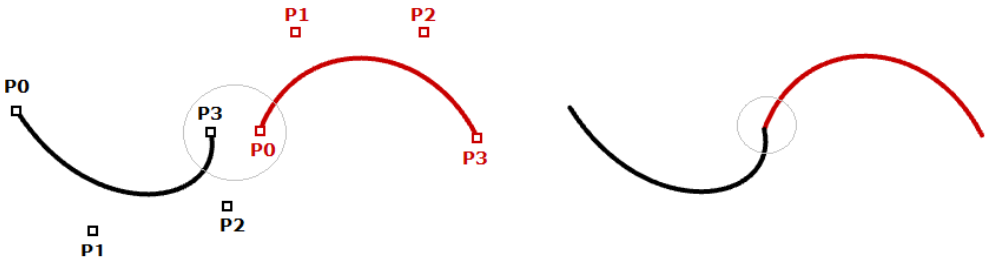


Figura (33): Dos segmentos Bézier comparten un punto.

Aunque las curvas Hermite emplean la misma cantidad de parámetros que las Bézier (4 para definir la curva), tenemos información adicional de la tangente que se puede compartir con el siguiente tramo para crear una curva de apariencia más suave con menos información, como se muestra a continuación.

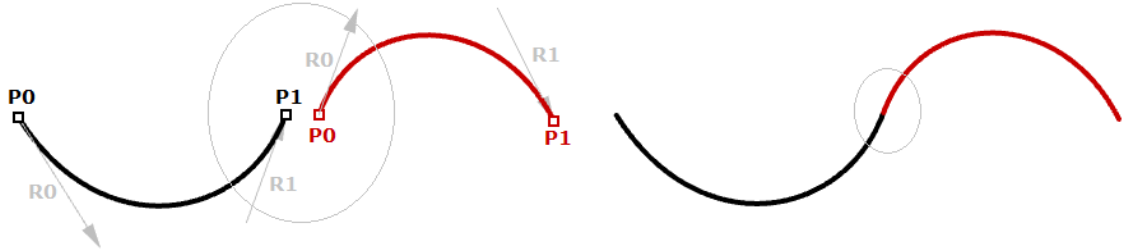


Figura (34): Dos segmentos Hermite comparten un punto y una tangente.

Para obtener curvas más suaves y continuas, hay una representación de curvas muy potente denominada Non Uniform Rational B-Spline[4] (NURBS). Los segmentos comparten más puntos de control para conseguir más suavidad con menos información.

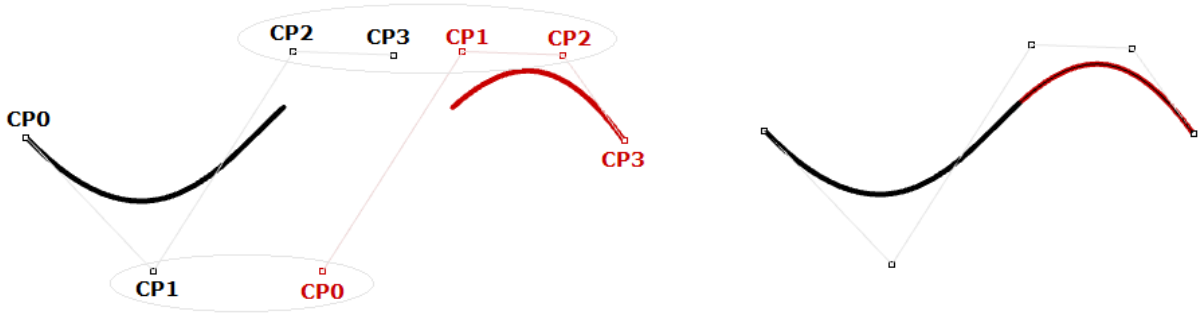


Figura (35): Dos segmentos NURBS de grado 3 comparten tres puntos de control.

Las curvas y superficies NURBS son la principal representación matemática que utiliza Rhino para representar la geometría. Las características y los componentes de las curvas NURBS se tratarán en detalle a lo largo de este capítulo.

[4] Wikipedia: NURBS.

Cálculo de curvas de Bézier cúbicas

Con el nombre de su inventor, Paul De Casteljau[5], el algoritmo de De Casteljau calcula las curvas Bézier utilizando un método iterativo. Los pasos del algoritmo se pueden resumir de la siguiente manera:

Entrada:

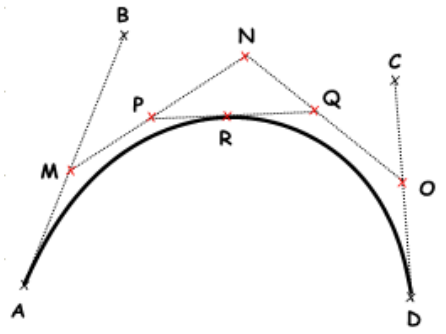
Cuatro puntos A, B, C, D definen una curva *t*, es cualquier parámetro dentro del dominio de la curva

Resultado:

Punto R en la curva que está en el parámetro *t*.

Solución:

1. Halle el punto M en el parámetro *t* de la línea AB.
2. Halle el punto N en el parámetro *t* de la línea BC.
3. Halle el punto O en el parámetro *t* de la línea CD.
4. Halle el punto P en el parámetro *t* de la línea MN.
5. Halle el punto Q en el parámetro *t* de la línea NO.
6. Halle el punto R en el parámetro *t* de la línea PQ.



Curvas NURBS

NURBS es un método exacto de representación matemática de curvas muy intuitivo a la hora de editar. Es fácil representar curvas de forma libre utilizando NURBS y la estructura de control hace que su edición sea fácil y predecible.

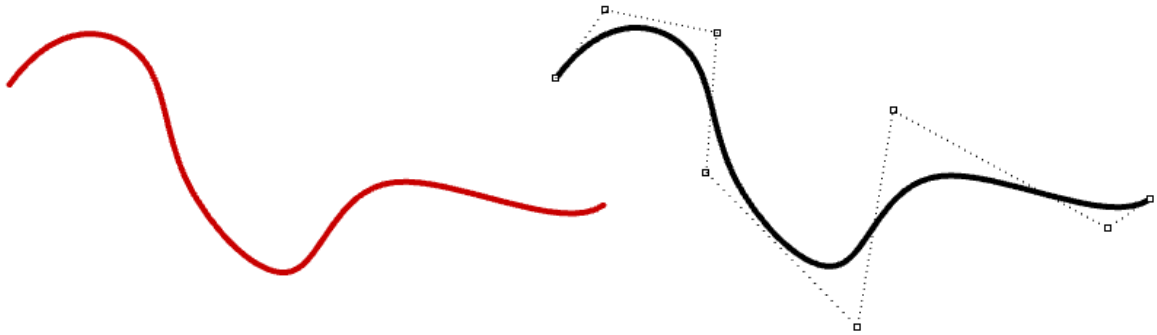


Figura (36): B-splines racionales no uniformes y su estructura de control.

[5] Wikipedia: algoritmo de De Casteljau

Existen multitud de libros y referencias para quien le interese investigar más en profundidad sobre las NURBS. Sin embargo, es necesario tener conocimientos básicos de las NURBS para poder utilizar un modelador NURBS de un modo eficaz. Una curva NURBS viene definida por cuatro atributos principales: *grado*, *puntos de control*, *nodos* y *reglas de cálculo*.

Grado

El grado de la curva es un número entero positivo. Rhino permite trabajar con cualquier grado empezando po el 1. Los grados 1, 2, 3 y 5 son los más útiles, mientras que los grados 4 y los superiores a 5 no se usan mucho en el mundo real. A continuación se muestran algunos ejemplos de curvas y su grado:

Las **líneas** y las **polilíneas** son curvas NURBS de grado 1.

Los **círculos** y las **elipses** son curvas NURBS de grado 2.

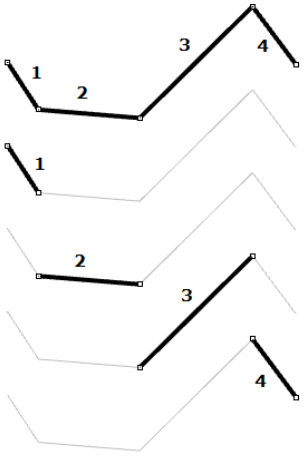
Las **curvas** forma libre normalmente se representan como curvas NURBS de grado 3 o 5.

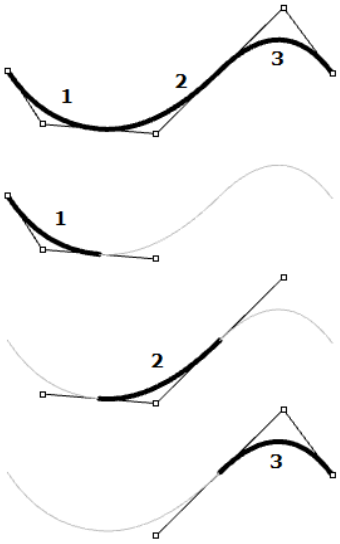
Puntos de control

Los puntos de control de una curva NURBS son una lista de puntos de grado 1 como mínimo. La manera más común de modificar la forma de una curva NURBS es moviendo sus puntos de control.

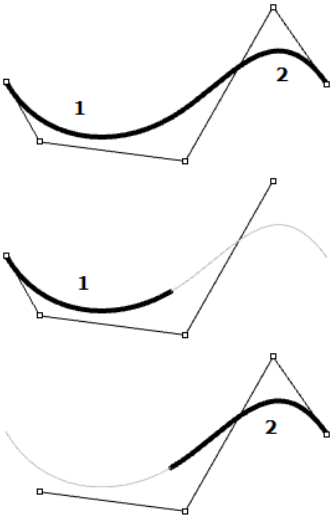
El número de puntos de control que afectan a cada segmento de una curva NURBS viene definido por el grado de la curva. Por ejemplo, cada segmento de una curva de grado 1 solo se ve afectado por los dos puntos de control finales. En una curva de grado 2, cada segmento se ve afectado por tres puntos de control, y así sucesivamente.

Los puntos de control de las curvas de grado 1 pasan por todos los puntos de control de la curva. En una curva NURBS de grado 1, dos puntos de control de grado 1 definen cada segmento. Con cinco puntos de control, la curva tiene cuatro segmentos.





Los círculos y las elipses son ejemplos de curvas de grado 2. En una curva NURBS de grado 2, tres puntos de control de grado 1 definen cada segmento. Con cinco puntos de control, la curva tiene tres segmentos.



Los puntos de control de las curvas de grado 3 no suelen tocar la curva, excepto en los puntos finales de las curvas abiertas. En una curva NURBS de grado 3, cuatro puntos de control de grado 1 definen cada segmento. Con cinco puntos de control, la curva tiene dos segmentos.

Peso de los puntos de control

Los puntos de control llevan asociado un número denominado *peso*. Con algunas excepciones, los pesos son números positivos. Cuando todos los puntos de control tienen el mismo peso, normalmente 1, la curva se denomina no racional. Los pesos son como la cantidad de gravedad que tiene cada punto de control. Cuanto mayor sea el peso relativo de un punto de control, más se acercará la curva a ese punto de control.

Cabe señalar que es mejor no cambiar los pesos de las curvas. Cambiar los pesos no suele dar el resultado deseado y puede presentar problemas de cálculo en operaciones como las intersecciones. El único buen motivo para usar curvas racionales es representar curvas que no se pueden dibujar de otra manera, como los círculos y las elipses.

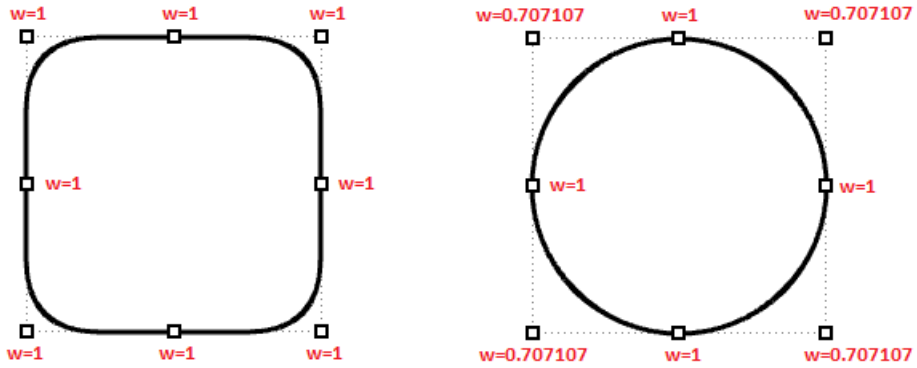


Figura (37): El efecto de cambiar el peso de los puntos de control en la curva resultante. La curva de la izquierda es no racional con pesos de puntos de control uniformes. El círculo de la derecha es una curva racional con puntos de control de esquina que tienen pesos inferiores a 1.

Nodos

Cada curva NURBS tiene una lista de números asociados a ella denominados *nodos* (o *vectores nodales*). Los nodos son algo más difíciles de comprender y manejar. Si utiliza un modelador 3D, no tendrá que definir manualmente los nodos para cada curva que cree.

Los nodos son valores de parámetros

Los nodos son una lista no decreciente de valores de parámetros que se encuentran dentro del dominio de la curva. En Rhino, hay más nodos de grado -1 que puntos de control. Es decir, el número de nodos es igual al número de puntos de control más el grado de curva menos 1:

|nodos| = |curvas| + grado -1

Normalmente, para curvas no periódicas, los nodos de primer grado son iguales al dominio mínimo y los nodos de último grado son iguales al dominio máximo.

Por ejemplo, los nodos de una curva NURBS de grado 3 abierta con 7 puntos de control y un dominio entre 0 y 4 serían así: <0, 0, 0, 1, 2, 3, 4,4, 4>.

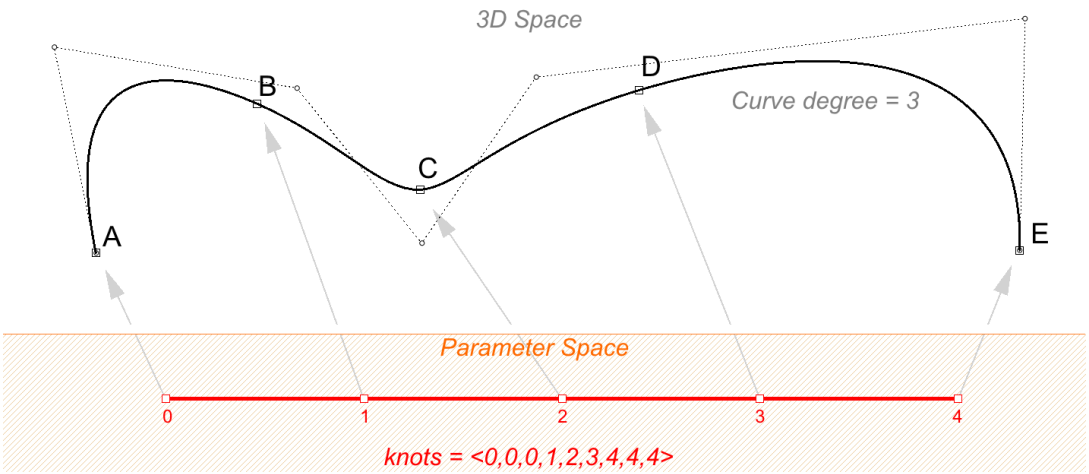


Figura (38): Hay más nodos de grado 1 que puntos de control. Si el número de puntos de control es de 7 y el grado de la curva es de 3, entonces el número de nodos es de 9. Los valores de los nodos son parámetros que suelen ser puntos en la curva 3D.

Escalar una lista de nodos no afecta a la curva 3D. Si cambia el dominio de la curva en el ejemplo anterior de "0 a 4" a "0 a 1", la lista de nodos se escala, pero la curva 3D no cambia.

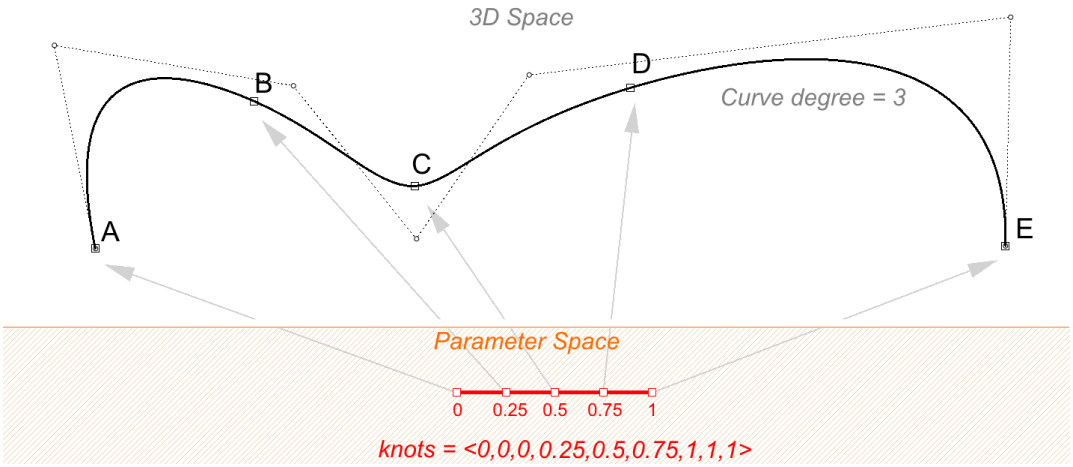


Figura (39): Escalar la lista de nodos no cambia la curva 3D.

Un nodo cuyo valor aparece solo una vez se denomina nodo simple. Los nodos interiores suelen ser simples, como en los dos ejemplos anteriores.

Multiplicidad de nodos

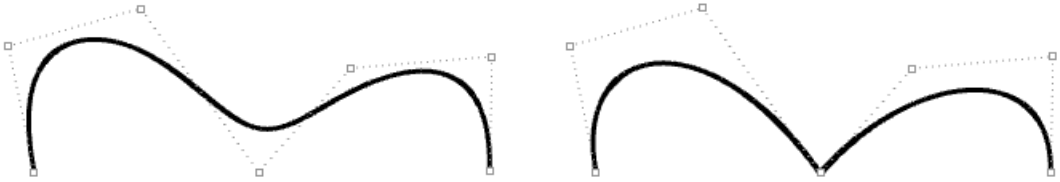
La multiplicidad de un nodo es el número de veces que se encuentra en la lista de nodos. La multiplicidad de un nodo no puede ser mayor que el grado de la curva. La multiplicidad nodal se utiliza para controlar la continuidad en el correspondiente punto de la curva.

Nodos de multiplicidad total

Un nodo de multiplicidad total tiene una multiplicidad igual al grado de la curva. En un nodo de multiplicidad total existe un punto de control que le corresponde, y la curva pasa por este punto.

Por ejemplo, las curvas ancladas tienen nodos de multiplicidad total en los finales de la curva. Esta es la razón por la que los puntos de control finales coinciden con los puntos finales de la curva. Los nodos interiores de multiplicidad total generan un punto de torsión en el punto correspondiente de la curva.

Por ejemplo, las dos curvas siguientes son de grado 3 y tienen el mismo número y ubicación de puntos de control. Sin embargo, tienen diferentes nodos y su forma también es diferente. La multiplicidad total obliga a la curva a pasar por el punto de control asociado.



(A): **nodos = <0,0,0,1,2,3,4,4,4>** (B): **nodos = <0,0,0,2,2,2,4,4,4>**

Figura (40) (A): Las curvas ancladas tienen nodos de multiplicidad total en el inicio y el final que son iguales al grado de la curva (3 en este caso). El resto de nodos son simples. (B): un nodo de multiplicidad total en el medio crea un punto de torsión y la curva se ve obligada a pasar por el punto de control asociado.

Nodos uniformes

Una lista uniforme de nodos en curvas abiertas satisface la siguiente condición:

Los nodos comienzan con un nodo de multiplicidad total, seguido de nodos simples y terminan con otro nodo de multiplicidad total. Los valores son crecientes a intervalos regulares. Esto es típico de las curvas abiertas (ancladas). Las curvas cerradas periódicas funcionan de manera diferente, como veremos más adelante.

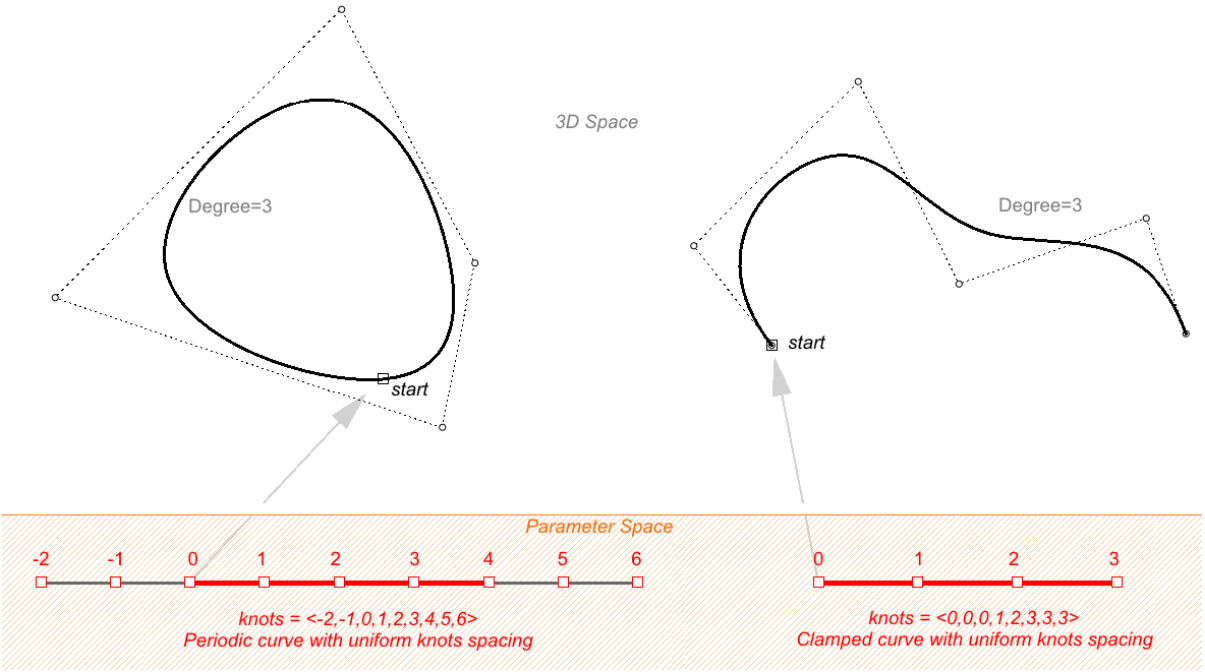


Figura (41): "Lista de nodos uniformes" significa que el espaciado entre los nodos es constante, con la excepción de las curvas ancladas, que pueden tener nodos de multiplicidad total al inicio y al final, y aún así considerarse uniformes. La curva izquierda es periódica (cerrada sin punto de torsión) y la derecha está anclada (abierta).

Nodos no uniformes

Las curvas NURBS pueden tener un espaciado no uniforme entre los nodos. Esto puede ayudar a controlar la curvatura a lo largo de la curva para crear curvas más suaves. Vea el siguiente ejemplo de interpolación a través de puntos usando una lista de nodos no uniformes a la izquierda y uniformes a la derecha. En general, si el espaciado de nodos de una curva NURBS es proporcional al espaciado entre los puntos de control, entonces la curva es más suave.

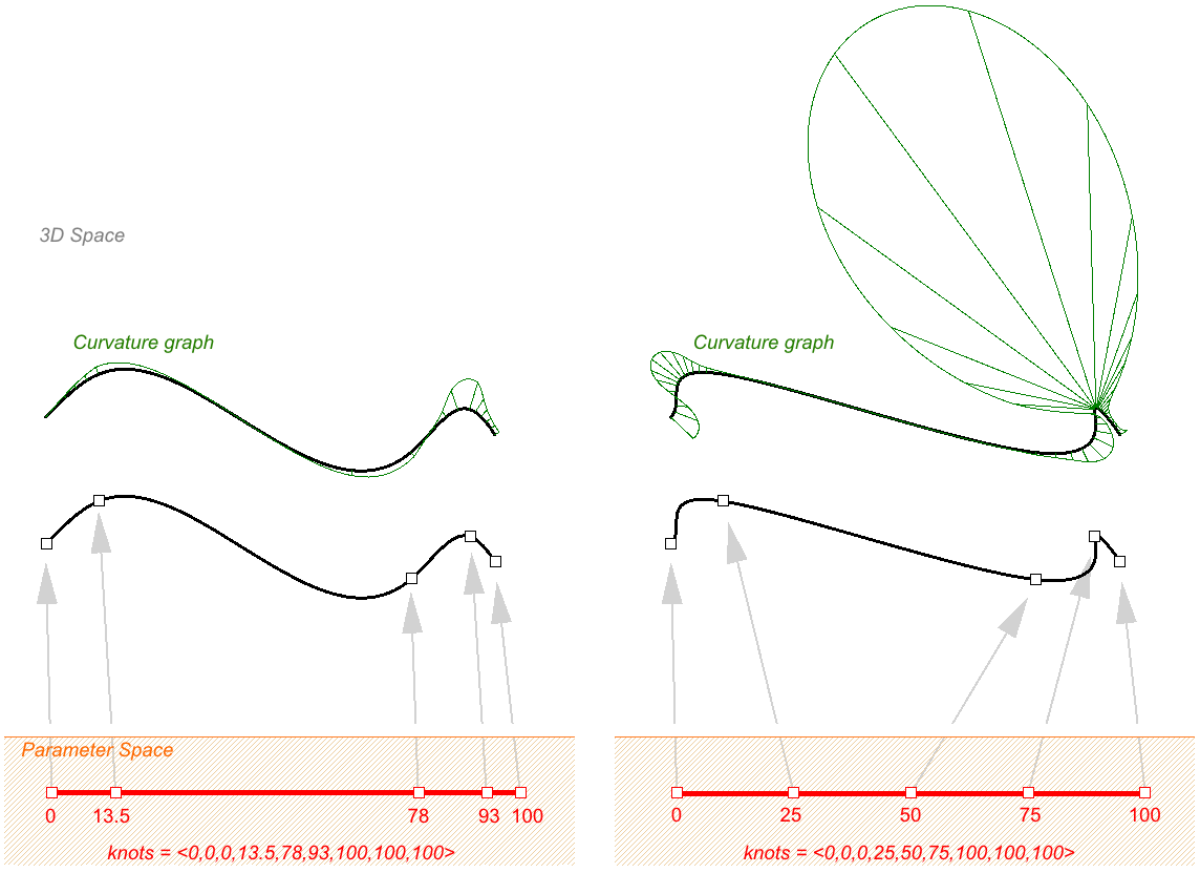


Figura (42): Una lista de nodos no uniformes puede ayudar a producir curvas más suaves. La curva de la izquierda se interpola a través de puntos con nodos no uniformes y produce una curvatura más suave. La curva de la derecha se interpola a través de los mismos puntos pero obliga a tener un espaciado uniforme de nodos, con lo cual la curva resultante no es tan suave.

Un ejemplo de curva no uniforme y racional es un círculo NURBS. La siguiente curva es de grado 2 con 9 puntos de control y 10 nodos. El dominio es 0-4 y el espaciado se alterna entre 0 y 1.

nodos = <0,0,1,1,2,2,3,3,4,4> --- (multiplicidad total en los nodos interiores)

espaciado entre nodos = [0,1,0,1,0,1,0,1,0] --- (no uniforme)

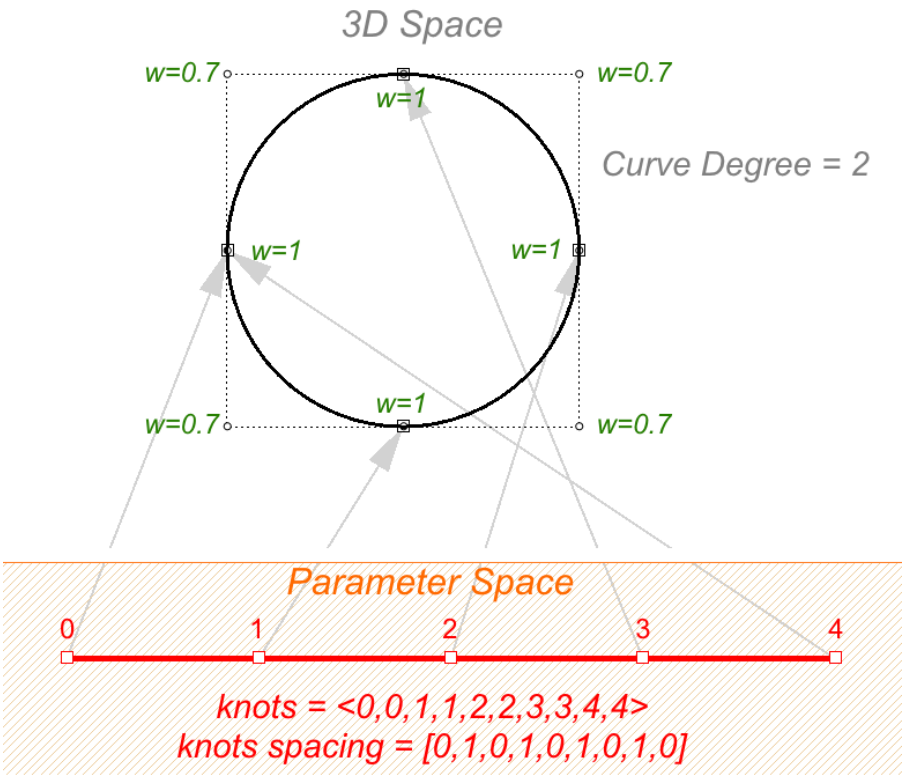


Figura (43): una aproximación NURBS de un círculo es una NURBS racional y no uniforme.

Regla de cálculo

La regla de cálculo utiliza una fórmula matemática que coge un número dentro del dominio de la curva y asigna un punto. La fórmula tiene en cuenta el grado, los puntos de control y los nodos.

Empleando esta fórmula, las funciones de curva especializadas pueden tomar un parámetro de curva y producir el punto correspondiente en esa curva. Un parámetro es un número que se encuentra dentro del dominio de la curva. Los dominios son normalmente crecientes y consisten en dos números: el parámetro mínimo del dominio (se conoce como t0) suele ser el punto inicial de la curva y el parámetro máximo (t1) suele ser el final de la curva.

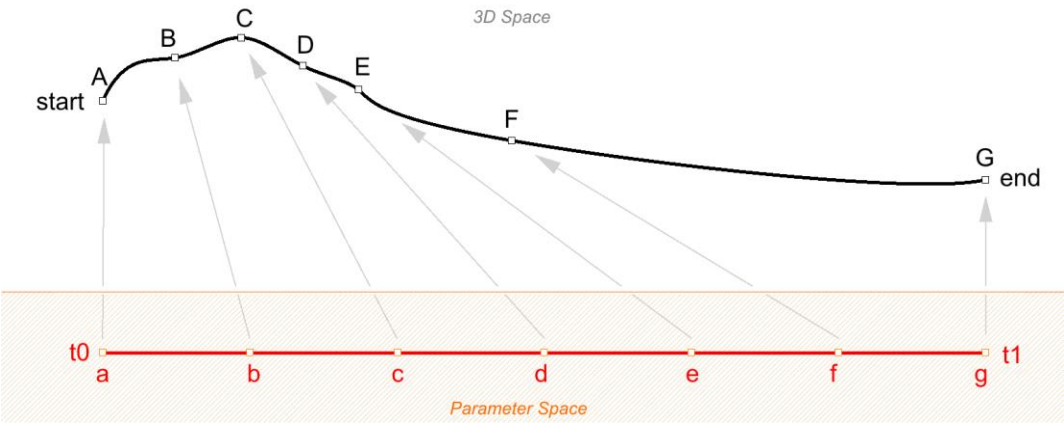


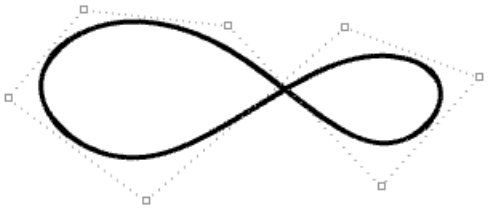
Figura (44): Parámetros (a, b, c...) que se traducen en puntos en la curva 3D (A, B, C, …). Los parámetros mínimo y máximo (t0 y t1) suelen ser los puntos inicial y final de la curva 3D.

Características de las curvas NURBS

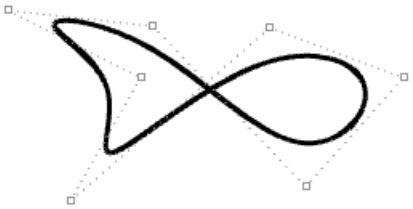
Para generar una curva NURBS, necesitará la siguiente información:

- Dimensión, normalmente 3
- Grado, (algunas veces se utiliza el *orden*, que es el grado+1)
- Puntos de control (lista de puntos)
- Peso de puntos de control (lista de números)
- Nodos (lista de números)

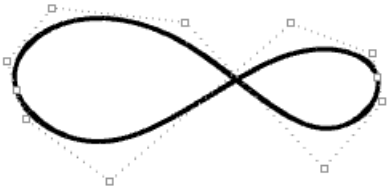
Cuando se crea una curva, es necesario al menos definir el grado y la ubicación de los puntos de control. El resto de la información necesaria para crear curvas NURBS se puede generar automáticamente. Si se selecciona un punto final para que coincida con el punto inicial, normalmente se crea una curva cerrada suave periódica. La siguiente tabla muestra ejemplos de curvas abiertas y cerradas:

Curva abierta de grado 1. La curva pasa por todos los puntos de control.	
Curva abierta de grado 3. Ambos finales de la curva coinciden con los puntos de control finales.	
Curva periódica cerrada de grado 3. La costura de la curva no pasa por ningún punto de control.	

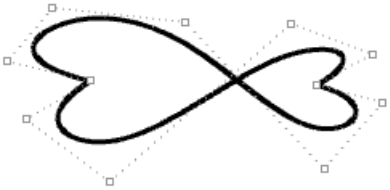
Mover los puntos de control de una curva periódica no repercute en la suavidad de la curva.



Las puntos de torsión se crean cuando la curva se ve obligada a pasar por algunos puntos de control.



Mover los puntos de control de una curva no periódica no garantiza una continuidad uniforme de la curva, pero permite un mayor control sobre el resultado.



Curvas NURBS ancladas vs. periódicas

Los puntos finales de las curvas cerradas ancladas coinciden con los puntos de control finales. Las curvas periódicas son curvas cerradas suaves. La mejor manera de comprender las diferencias entre las dos es comparando los puntos de control y los nodos.

El siguiente ejemplo es una curva NURBS no racional, anclada y abierta. Esta curva tiene cuatro puntos de control, nodos uniformes de multiplicidad total en los nodos inicial y final y los pesos de todos los puntos de control iguales a 1.

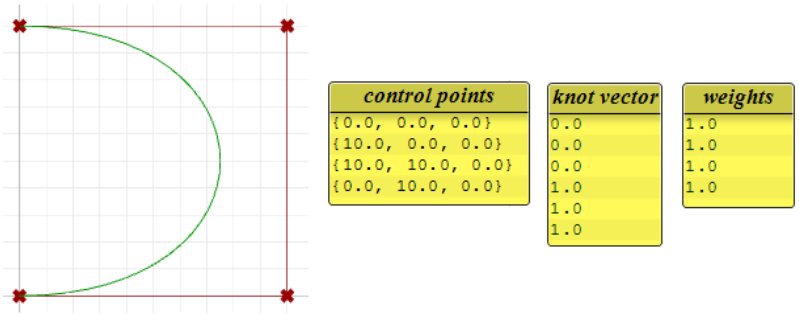
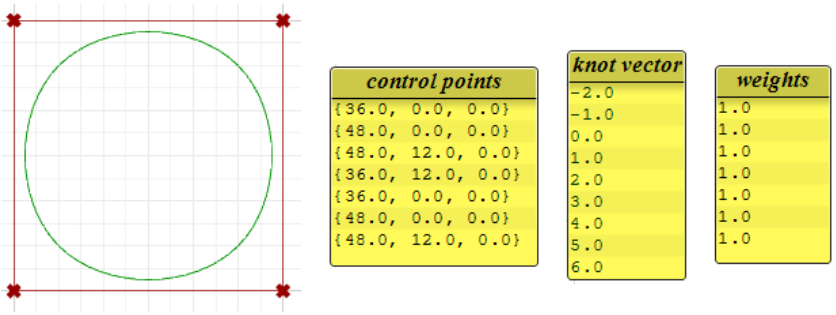


Figura (45): Curva NURBS no racional abierta de grado 3.

La siguiente curva circular es un ejemplo de una curva NURBS periódica cerrada de grado 3. También es no racional porque todos los pesos son iguales. Tenga en cuenta que las curvas periódicas necesitan más puntos de control con pocas superposiciones. Los nodos también son simples.



control points
{36.0, 0.0, 0.0}
{48.0, 0.0, 0.0}
{48.0, 12.0, 0.0}
{36.0, 12.0, 0.0}
{36.0, 0.0, 0.0}
{48.0, 0.0, 0.0}
{48.0, 12.0, 0.0}

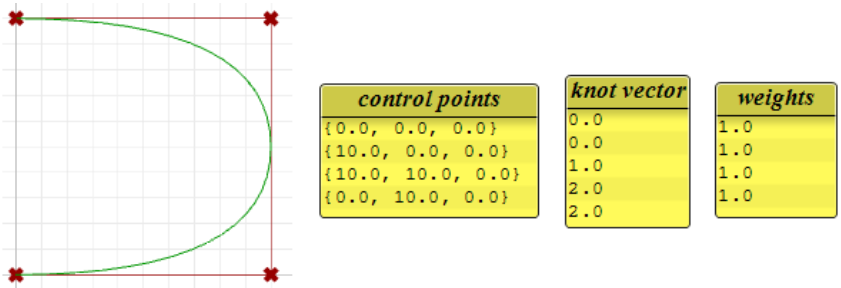
knot vector
-2.0
-1.0
0.0
1.0
2.0
3.0
4.0
5.0
6.0

weights
1.0
1.0
1.0
1.0
1.0
1.0
1.0

Figura (46): Curva NURBS (periódica) cerrada de grado 3.

Observe que la curva periódica convirtió los cuatro puntos de entrada en siete puntos de control (grado+4), mientras que la curva anclada solo utilizó los cuatro puntos de control. Los nodos de la curva periódica utilizan solo nodos simples, mientras que los nodos de inicio y final de la curva anclada tienen multiplicidad total, lo que obliga a la curva a pasar por los puntos de control de inicio y final.

Si establecemos el grado de los ejemplos anteriores a 2 en lugar de 3, los nodos se vuelven más pequeños y el número de puntos de control de las curvas periódicas cambia.

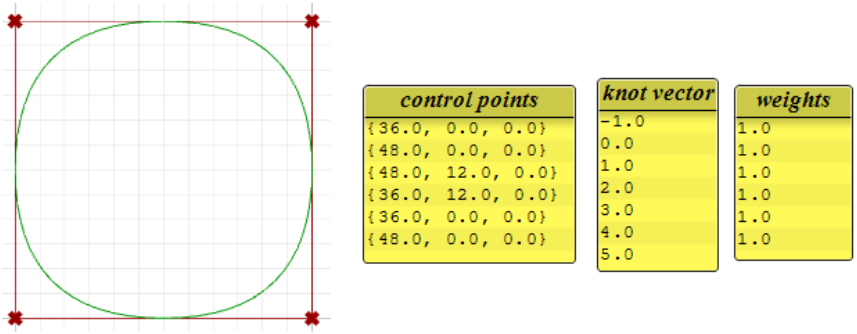


control points
{0.0, 0.0, 0.0}
{10.0, 0.0, 0.0}
{10.0, 10.0, 0.0}
{0.0, 10.0, 0.0}

knot vector
0.0
0.0
1.0
2.0
2.0

weights
1.0
1.0
1.0
1.0

Figura (47): Curva NURBS abierta de grado 2.



control points
{36.0, 0.0, 0.0}
{48.0, 0.0, 0.0}
{48.0, 12.0, 0.0}
{36.0, 12.0, 0.0}
{36.0, 0.0, 0.0}
{48.0, 0.0, 0.0}

knot vector
-1.0
0.0
1.0
2.0
3.0
4.0
5.0

weights
1.0
1.0
1.0
1.0
1.0
1.0

Figura (48): Curva NURBS (periódica) cerrada de grado 2.

Pesos

Los pesos de los puntos de control de una curva NURBS uniforme se establecen en 1, pero este número puede variar en curvas NURBS racionales. El siguiente ejemplo muestra el efecto de variar los pesos de los puntos de control.

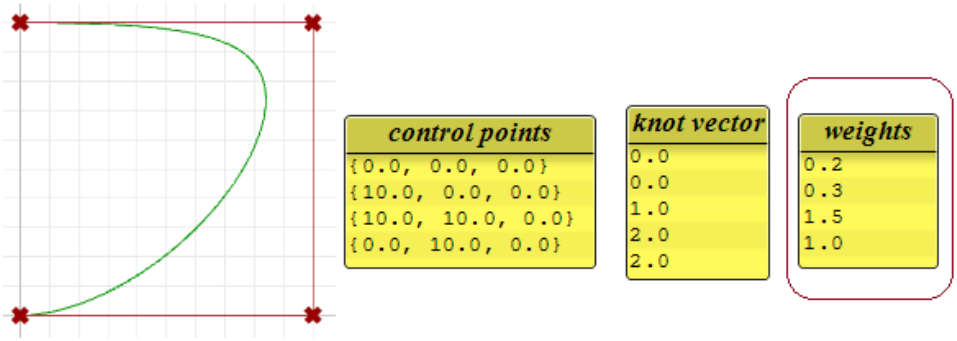


Figura (49): Pesos en una curva NURBS abierta.

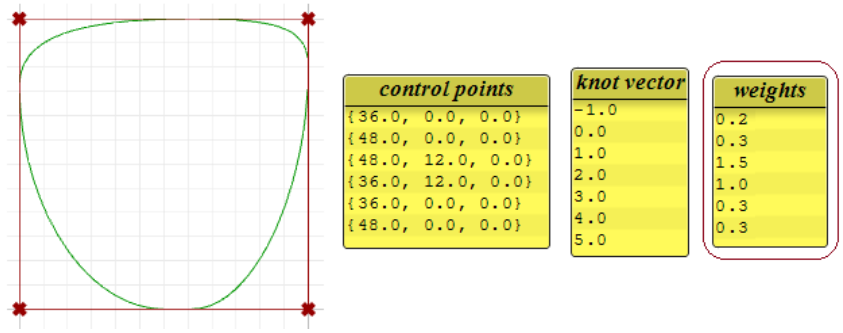


Figura (50): Pesos en una curva NURBS cerrada.

Cálculo de curvas NURBS

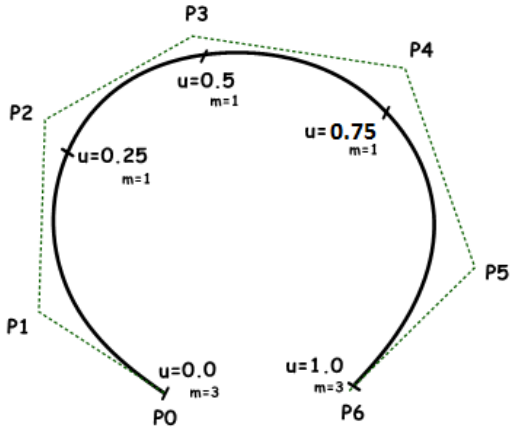
El algoritmo de De Boor[6], que lleva el nombre de su inventor, Carl de Boor, es una generalización del algoritmo de De Casteljau para curvas de Bézier. Es estable numéricamente, y su uso está muy extendido para calcular puntos en curvas NURBS dentro de las aplicaciones 3D. A continuación se muestra un ejemplo para calcular un punto en una curva NURBS de grado 3 utilizando el algoritmo de De Boor.[7]

Entrada:

Siete puntos de control P_0 a P_6

Nodos:

u0 = 0.0
u1 = 0.0
u2 = 0.0
u3= 0.25
u4 = 0.5
u5 = 0.75
u6 = 1.0
u7 = 1.0
u8 = 1.0



Resultado:

[6] Wikipedia: algoritmo de De Boor.
[7] Michigan Tech, Department of Computer Science, De Boor's algorithm

Punto de la curva que está en u=0.4

Solución:

1. Calcule los coeficientes para la primera iteración:

   Ac = $(u - u_1) / (u_{1+3} - u_1) = 0.8$

   Bc = $(u - u_2) / (u_{2+3} - u_2) = 0.53$

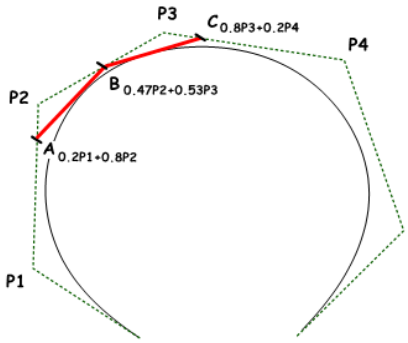
   Cc = $(u - u_3) / (u_{3+3} - u_3) = 0.2$

2. Calcule los puntos usando datos de los coeficientes:

   A = $0.2\mathbf{P_1} + 0.8\mathbf{P_2}$

   B = $0.47\ \mathbf{P_2} + 0.53\ \mathbf{P_3}$

   C = $0.8\ \mathbf{P_3} + 0.2\ \mathbf{P_4}$



3. Calcule los coeficientes para la segunda iteración:

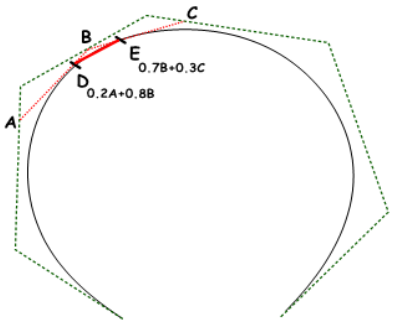
   Dc = $(u - u_2) / (u_{2+3-1} - u_2) = 0.8$

   Ec = $(u - u_3) / (u_{3+3-1} - u_3) = 0.3$

4. Calcule los puntos usando datos de los coeficientes:

   D = $0.2\mathbf{A} + 0.8\mathbf{B}$

   E = $0.7\mathbf{B} + 0.3\mathbf{C}$

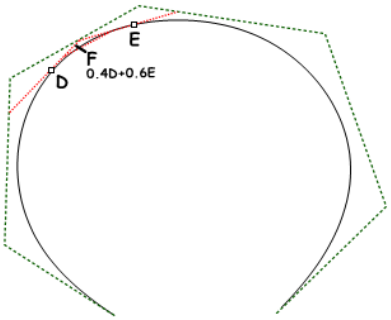


5. Calcule el último coeficiente:

   Fc = $(u - u_3)/ (u_{3+3-2} - u_3) = 0.6$

   Encuentre el punto en la curva en el parámetro u=0.4,

   F= $0.4\mathbf{D} + 0.6\mathbf{E}$



Continuidad geométrica de curva

La continuidad es un concepto importante en el modelado 3D. La continuidad es importante para lograr la suavidad visual y para obtener una luz y un flujo de aire suaves.

La siguiente tabla muestra varias continuidades y sus definiciones:

G0 (continuidad de posición)	Dos segmentos de curva unidos
G1 (continuidad de tangencia)	La dirección de la tangente en el punto de unión es la misma para ambos segmentos de la curva
G2 (continuidad de curvatura)	Tanto las curvaturas como las tangentes coinciden para ambos segmentos de curva en el punto final común
GN	Las curvas coinciden con un orden superior

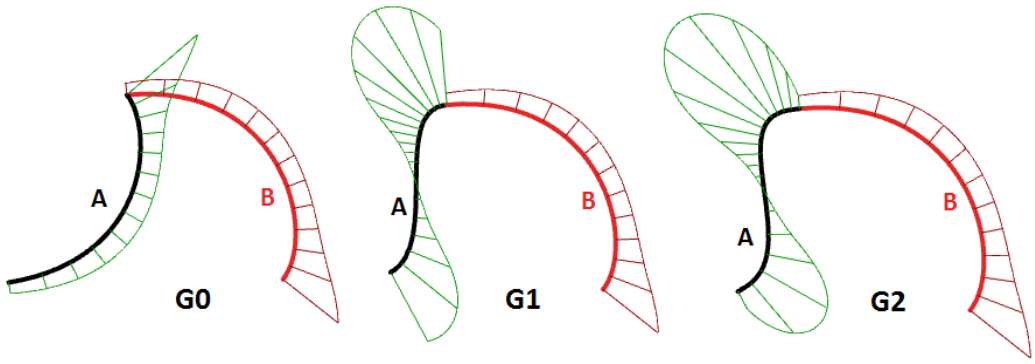


Figura (51): Continuidad de la curva con análisis del gráfico de curvatura.

Curvatura de curva

La curvatura es un concepto ampliamente utilizado en el modelado de superficies y curvas 3D. La curvatura se define como *el cambio en la inclinación de la tangente de una curva por unidad de longitud de arco*. Para un círculo o esfera, es la inversa del radio y es constante en todo el dominio.

En cualquier punto en una curva del plano, la línea que mejor se aproxima a la curva que atraviesa este punto es la línea tangente. También podemos encontrar el círculo más aproximado que atraviese este punto y sea tangente a la curva. La inversa del radio de este círculo es la curvatura de la curva en este punto.

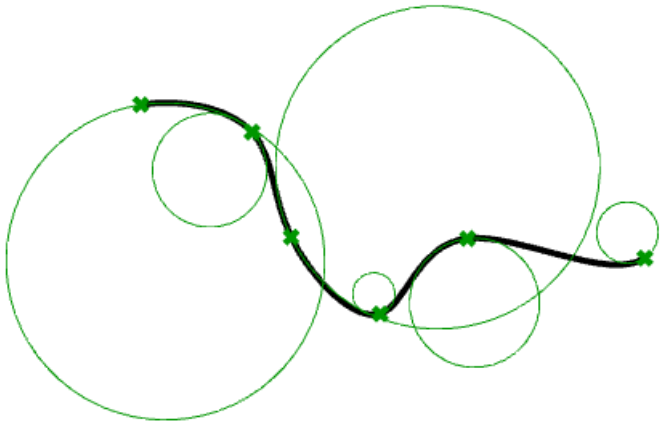


Figura (52): Curvatura de la curva en diferentes puntos.

El círculo más aproximado puede estar situado a la izquierda de la curva o a la derecha. Si tenemos esto en cuenta, podemos establecer una convención, como dar un símbolo positivo a la curvatura si el círculo se encuentra a la izquierda y negativo si el círculo se encuentra a la derecha de la curva. Esto se denomina *curvatura señalada*. Los valores de curvatura de las curvas unidas indican la continuidad entre estas curvas.

Superficies paramétricas

Parámetros de superficie

Una superficie paramétrica es una función de dos parámetros independientes (generalmente *u, v)* sobre algún dominio bidimensional. Tomemos como ejemplo un plano. Si tenemos un punto P en el plano y dos vectores no paralelos en el plano, **a** y b, entonces podemos definir una ecuación paramétrica del plano en términos de los dos parámetros *u* y *v* de la siguiente manera:

$$P = P' + u * \mathbf{a} + v * \mathbf{b}$$

Donde:

P' es un punto conocido en el plano

a es el primer vector en el plano

b es el primer vector en el plano

u es el primer parámetro

v es el primer parámetro

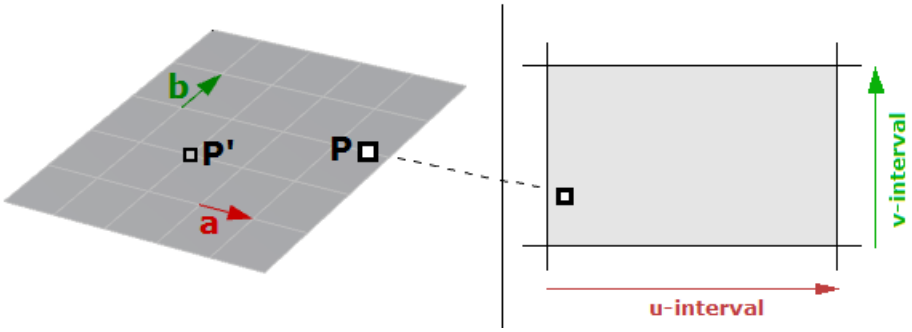


Figura (53): Rectángulo de parámetros de un plano.

Otro ejemplo es la esfera. La ecuación cartesiana de una esfera centrada en el origen con radio R es

$x^2 + y^2 + z^2 = R^2$

Eso significa que para cada punto hay tres variables (x, y, z), lo cual no es útil para una representación paramétrica que requiere solo dos variables. Sin embargo, en el sistema de coordenadas esféricas, cada punto utiliza los tres valores:

r: distancia radial entre el punto y el origen

θ: ángulo desde el eje x en el plano xy

ø: ángulo desde el eje z y el punto

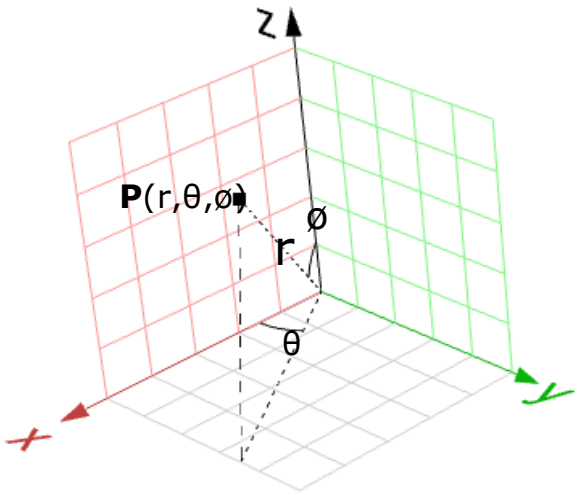


Figura (54): Sistema de coordenadas esféricas.

Se puede obtener una conversión de puntos de coordenadas esféricas a cartesianas de la siguiente manera:

x = ***r*** * sin(ø) * cos(θ)

y = ***r*** * sin(ø) * sin(θ)

z = ***r*** * cos (ø)

Donde:

r es la distancia desde el origen ≥ 0

θ va de 0 a 2π

ø va de 0 aπ

Puesto que **r** es constante en una superficie de esfera, nos quedan solo dos variables y, por tanto, podemos usar lo indicado anteriormente para crear una representación paramétrica de una superficie de esfera:

u = θ

v = ø

De manera que:

x = **r** * sin(v) * cos(u)

y = **r** * sin(v) * sin(u)

z = **r** * cos(v)

Donde (u, v) está dentro del dominio (2 π, π)

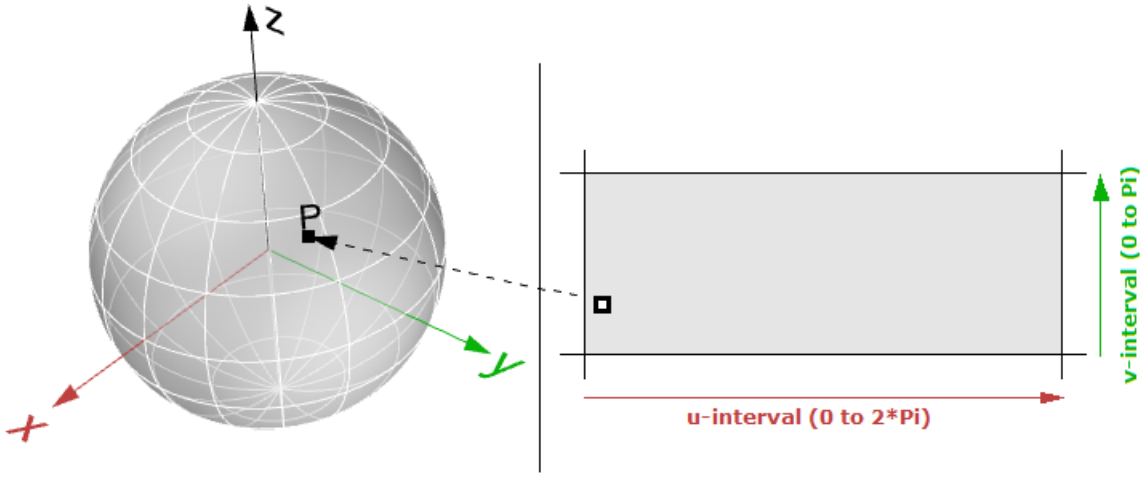


Figura (55): Rectángulo de parámetros de una esfera.

La superficie paramétrica sigue la forma general:

x = x(u,v)

y = y(u,v)

z = z(u,v)

Donde:

u y v son los dos parámetros dentro del dominio o región de superficie.

Dominio de superficie

Un dominio de superficie se define como el rango de parámetros (u, v) que se traducen en un punto 3D en esa superficie. El dominio en cada dimensión (u o v) generalmente se describe como dos números reales (u_min a u_max) y (v_min a v_max).

El cambio de un dominio de superficie se conoce como *reparametrizar* la superficie. Un dominio creciente significa que el valor mínimo del dominio apunta al punto mínimo de la superficie. Los dominios suelen aumentar, pero no siempre.

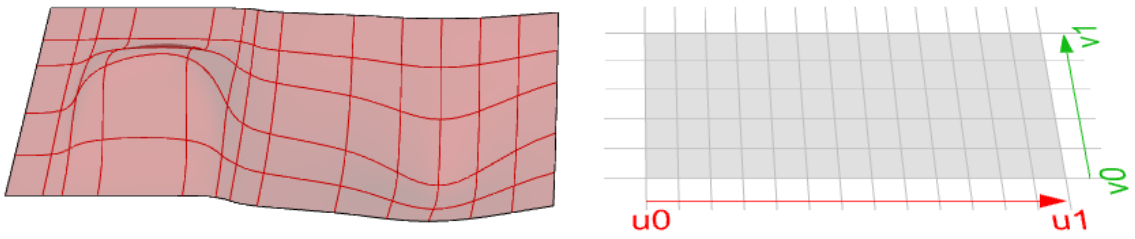


Figura (56): Superficie NURBS en el espacio de modelado 3D (izquierda). Rectángulo de parámetros de superficie con dominio que se extiende desde u0 a u1 en la primera dirección y v0 a v1 en la segunda dirección (derecha).

Cálculo de superficies

Calcular una superficie en un parámetro que está dentro del dominio de la superficie da como resultado un punto que está en la superficie. Tenga en cuenta que el medio del dominio (midu, midv) puede no calcular necesariamente el punto medio de la superficie 3D. Además, el cálculo de los valores u y v que están fuera del dominio de la superficie no dará un resultado útil.

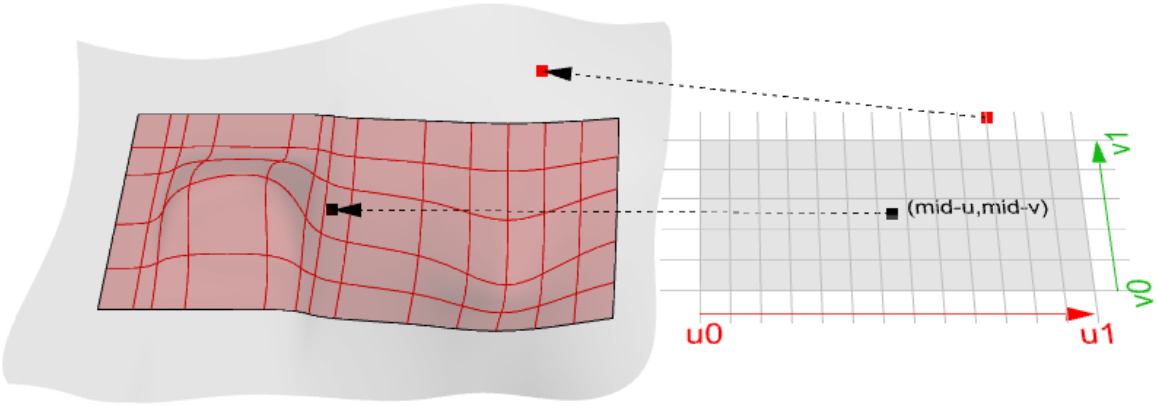


Figura (57): Cálculo de superficies.

Plano tangente de una superficie

El plano tangente a una superficie en un punto dado es el plano que toca la superficie en ese punto. La dirección z del plano tangente representa la dirección normal de la superficie en ese punto.

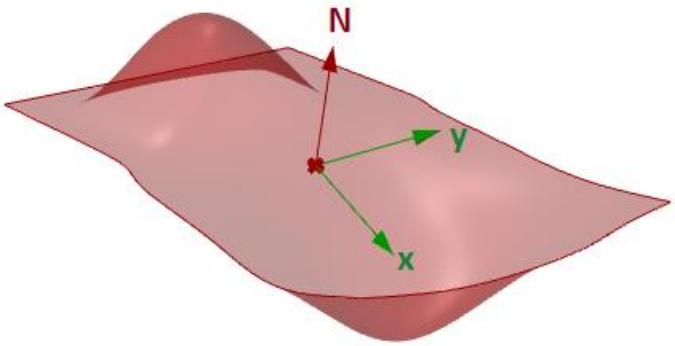


Figura (58): Vectores tangentes y normales a una superficie.

Continuidad geométrica de superficie

Muchos modelos no se pueden construir a partir de un parche de superficie. La continuidad entre los parches de superficies unidas es importante para la suavidad visual, el reflejo de la luz y el flujo de aire.

La siguiente tabla muestra varias continuidades y sus definiciones:

G0 (continuidad de posición)	Dos superficies unidas.
G1(continuidad de tangencia)	Las tangentes correspondientes de las dos superficies a lo largo de su borde de unión son paralelas en ambas direcciones u y v.
G2 (continuidad de curvatura)	Las curvaturas y las tangentes coinciden para ambas superficies en el borde común.
GN	Las superficies coinciden en un orden mayor.

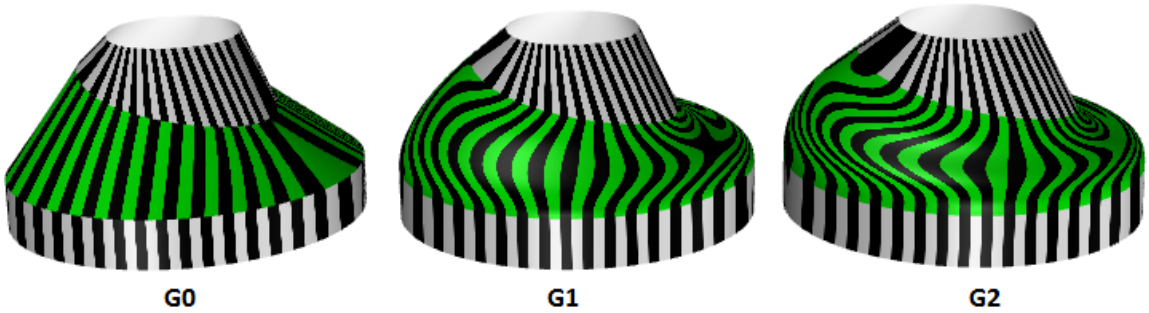


Figura (59): Comprobación de continuidad de superficie con análisis de cebra.

Curvatura de superficie

Para superficies, la curvatura normal es una generalización de la curvatura aplicada a las superficies. Dado un punto en la superficie y una dirección situada en el plano tangente de la superficie en ese punto, la curvatura de sección normal se calcula intersecando la superficie con el plano segmentado por el punto, la normal a la superficie en ese punto y la dirección. La curvatura de sección normal es la curvatura señalada de esta curva en el punto de interés.

Si miramos en todas las direcciones en el plano tangente a la superficie en nuestro punto y calculamos la curvatura normal en todas esas direcciones, habrá un valor máximo y un valor mínimo.

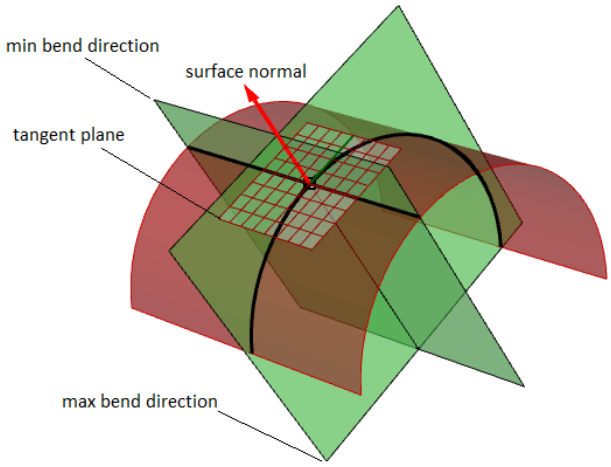


Figura (60): Curvaturas normales.

Curvaturas principales

Las curvaturas principales de una superficie en un punto son el mínimo y el máximo de las curvaturas normales en ese punto. Miden la cantidad máxima y mínima de curvatura de la superficie en ese punto. Las curvaturas principales se usan para calcular las curvaturas gaussianas y medias de la superficie.

Por ejemplo, en una superficie cilíndrica, no hay ninguna curvatura en la dirección lineal (la curvatura es igual a cero) mientras que la curvatura máxima se produce cuando se interseca con un plano paralelo a las caras de los extremos (la curvatura es igual a 1/radio). Esos dos extremos forman las curvaturas principales de esa superficie.

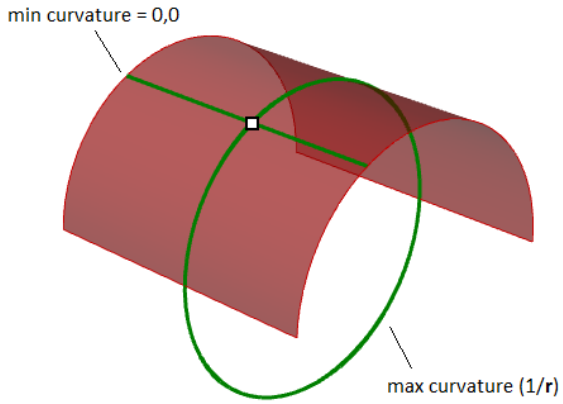
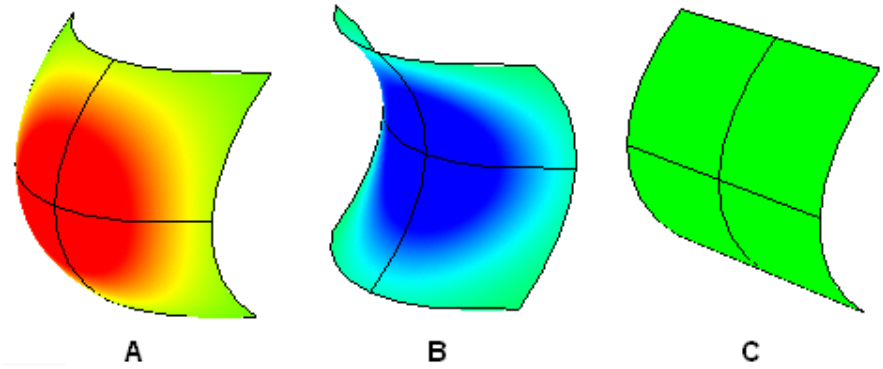


Figura (61): Las curvaturas principales de un punto en una superficie son las curvaturas mínima y máxima en ese punto.

Curvatura gaussiana

La curvatura gaussiana de una superficie en un punto es el producto de las curvaturas principales en ese punto. El plano tangente de cualquier punto con curvatura gaussiana positiva toca la superficie localmente en un solo punto, mientras que el plano de cualquier punto con curvatura gaussiana negativa corta la superficie.



R: curvatura positiva cuando la superficie tiene forma de cuenco.

B: curvatura negativa cuando la superficie tiene forma de silla de montar.

C: curvatura cero cuando la superficie es plana en al menos una dirección (plano, cilindro).

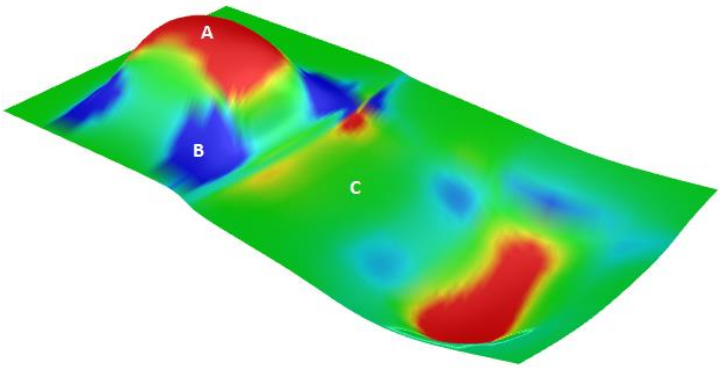


Figura (62): Curvatura gaussiana de superficie.

Curvatura media

La curvatura media de una superficie en un punto la mitad de las sumas de las curvaturas principales en ese punto. Cualquier punto con un curvatura media de cero tiene una curvatura gaussiana negativa o de cero.

Las superficies con una curvatura media de cero en todas partes son superficies mínimas. Los procesos físicos que pueden ser modelados por superficies mínimas incluyen la formación de capas de jabón que se extienden en objetos fijos, como bucles de estructura alámbrica. Una capa de jabón no se deforma por la presión del aire (que es igual en ambos lados) y es libre de minimizar su área. Por el contrario, una burbuja de jabón encierra una cantidad fija de aire y tiene presiones desiguales en el interior y el exterior. La curvatura media sirve para hallar áreas de cambio brusco en la curvatura de la superficie.

Las superficies con una curvatura media constante en todas partes a menudo se conocen como superficies de curvatura media constante (CMC). Las superficies CMC incluyen la formación de burbujas, tanto libres como asociadas a los objetos. Una burbuja de jabón, a diferencia de una simple capa de jabón, encierra un volumen y existe en equilibrio donde la presión ligeramente mayor dentro de la burbuja queda compensada por las fuerzas de superficie mínima de la misma burbuja.

Superficies NURBS

Las superficies NURBS son como una rejilla de curvas NURBS en dos direcciones. La forma de una superficie NURBS está definida por el número de puntos de control y el grado de la superficie en cada una de las dos direcciones (u y v). Las superficies NURBS sirven para almacenar y representar superficies de forma libre con un alto grado de precisión. Las ecuaciones matemáticas y los detalles de las superficies NURBS están fuera del alcance de este documento. Solo nos centraremos en las características que son más útiles para los diseñadores.

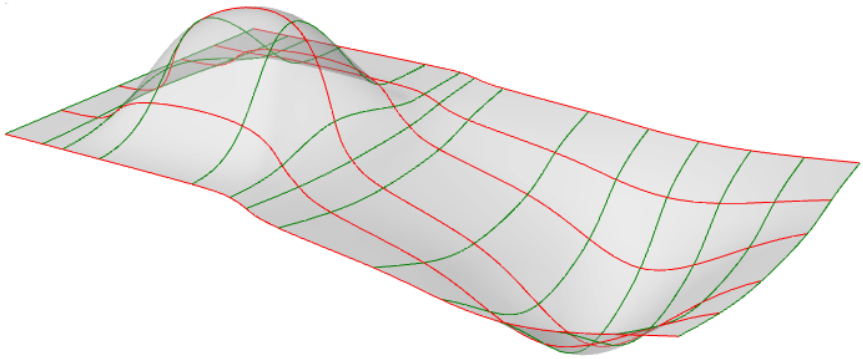


Figura (63): Superficie NURBS con isocurvas rojas en la dirección u e isocurvas verdes en la dirección v.

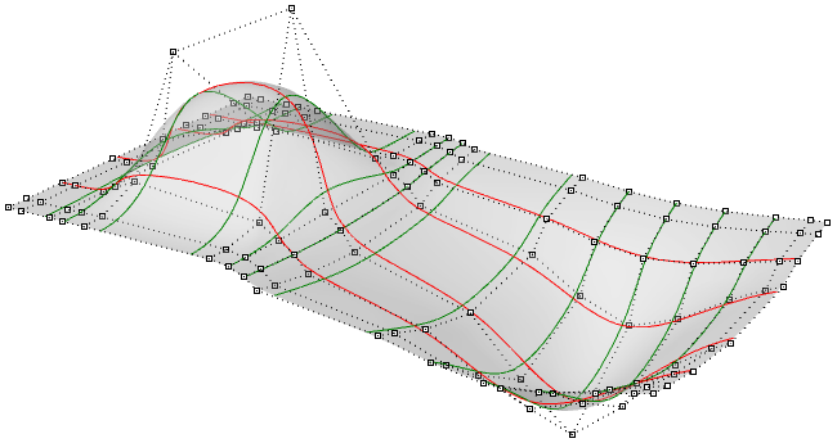


Figura (64): Estructura de control de una superficie NURBS.

Figura (65): Rectángulo de parámetros de una superficie NURBS.

Calcular parámetros a intervalos iguales en el rectángulo de parámetros 2D no se traduce en intervalos iguales en el espacio 3D en la mayoría de los casos.

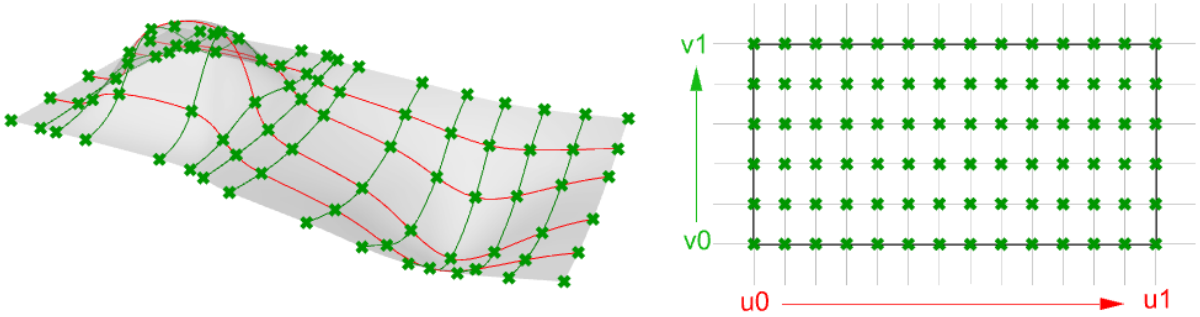


Figura (66): Cálculo de superficies.

Características de las superficies NURBS

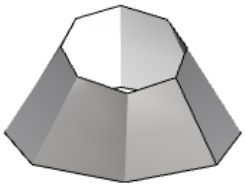
Las características de las superficies NURBS son muy similares a las de las curvas NURBS excepto en que tienen un parámetro adicional. Las superficies NURBS contienen la siguiente información:

- Dimensión, normalmente 3
- Grado en las direcciones u y v: (a veces se utiliza el *orden*, que es el grado 1)
- Puntos de control (puntos)
- Pesos de los puntos de control (números)
- Nodos (números)

Al igual que con las curvas NURBS, probablemente no necesitará conocer los detalles de cómo crear una superficie NURBS, ya que los modeladores 3D normalmente disponen de un buen conjunto de herramientas para ayudarle. Siempre se pueden reconstruir las superficies (y las curvas) a un nuevo grado y número de puntos de control. Una superficie puede ser abierta, cerrada o periódica. A continuación, se muestran algunos ejemplos de superficies:

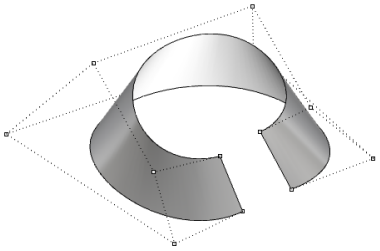
Superficie de grado 1 en ambas direcciones u y v.

Todos los puntos de control se encuentran en la superficie.



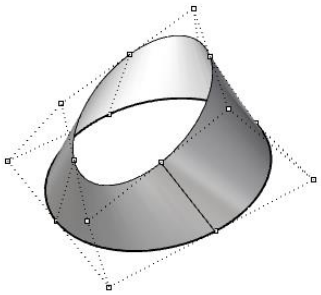
Superficie abierta de grado 3 en la dirección u y de grado 1 en la en la dirección v.

Las esquinas de la superficie coinciden con los puntos de control de las esquinas.

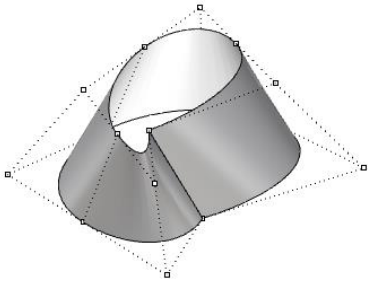


Superficie cerrada (no periódica) de grado 3 en la dirección u y de grado 1 en la en la dirección v.

Algunos puntos de control coinciden con la costura de la superficie.

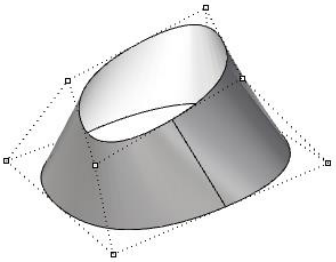


Al mover los puntos de control de una superficie cerrada (no periódica), se crea un punto de torsión y la superficie deja de ser suave.

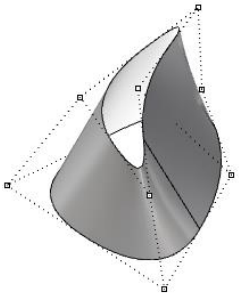


Superficie periódica de grado 3 en la dirección u y de grado 1 en la dirección v.

Los puntos de control de la superficie no coinciden con la costura de la superficie.



Mover los puntos de control de una superficie periódica no repercute en su suavidad ni crea puntos de torsión.



Singularidad en las superficies NURBS

Por ejemplo, si tiene un borde lineal de un plano simple y arrastra los dos puntos de control finales de un borde para que se superpongan (contraigan) en el medio, obtendrá un borde singular. Verá que las curvas isoparamétricas de la superficie convergen en el punto singular.

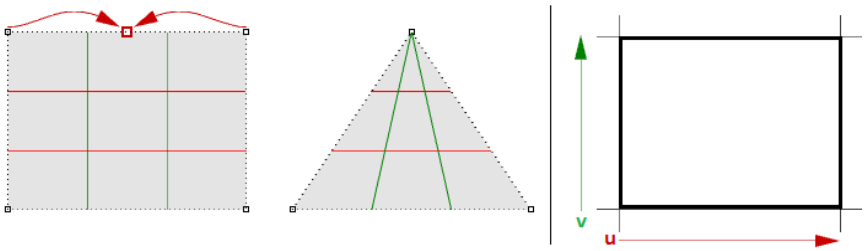


Figura (67): Contraiga dos puntos de esquina de una superficie NURBS rectangular para crear una superficie triangular con singularidad. El rectángulo de parámetros permanece rectangular.

La forma triangular anterior se puede crear sin singularidad. Puede recortar una superficie con una polilínea triangular. Si examina la estructura NURBS subyacente, verá que sigue siendo una forma rectangular.

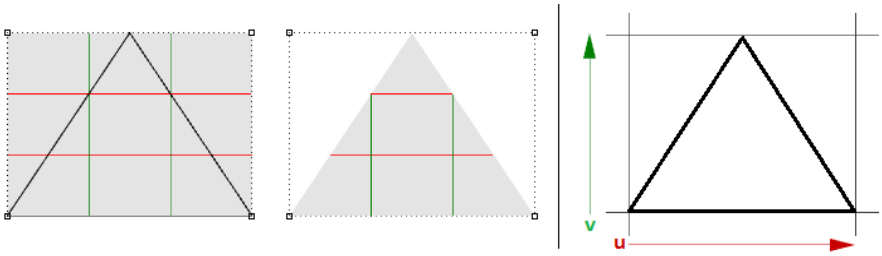


Figura (68): Recorte una superficie NURBS rectangular para crear una superficie triangular recortada.

Otros ejemplos comunes de superficies difíciles de generar sin singularidad son el cono y la esfera. La parte superior de un cono y los bordes superior e inferior de una esfera se contraen en un solo punto. Tanto si hay singularidad como si no la hay, el rectángulo de parámetros mantiene una zona más o menos rectangular.

Superficies NURBS recortadas

Las superficies NURBS pueden ser recortadas (*trimmed*) o no recortadas (*untrimmed*). Las superficies recortadas utilizan una superficie NURBS subyacente y curvas cerradas para recortar parte de esa superficie. Cada superficie tiene una curva cerrada que define el borde exterior (*bucle exterior*) y puede tener curvas interiores cerradas que no se intersecan para definir agujeros (*bucles internos*). Una superficie con un bucle exterior igual que el de su superficie NURBS subyacente y que no tiene agujeros se denomina superficie *no recortada*.

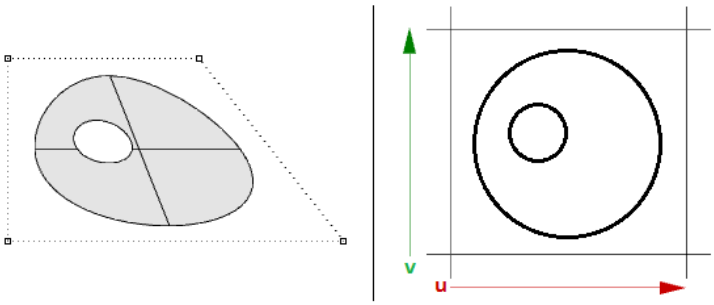


Figura (69): Superficie recortada en el espacio de modelado (izquierda) y en el rectángulo de parámetros (derecha).

Polisuperficies

Una polisuperficie está formada por dos o más superficies unidas (posiblemente recortadas). Cada superficie tiene su propia estructura, parametrización y direcciones de isocurva que no tienen por qué coincidir. Las polisuperficies se representan mediante lo que se denomina *boundary representation* (representación de bordes), abreviado *brep*. La estructura brep describe superficies, bordes y vértices con datos de los recortes y relaciones entre las diferentes partes. Las superficies recortadas también se representan mediante la estructura de datos brep.

Figura (70): Polisuperficies creadas a partir de superficies unidas con bordes comunes que se encuentran dentro de la tolerancia.

La brep es una estructura de datos que describe cada cara en términos de su superficie subyacente, bordes 3D que la rodean, vértices, recortes 2D del espacio de parámetros y relaciones con las caras adyacentes. Los objetos brep también se denominan *sólidos* cuando son cerrados (herméticos).

Un ejemplo de polisuperficie es una caja simple compuesta por seis superficies no recortadas unidas entre sí.

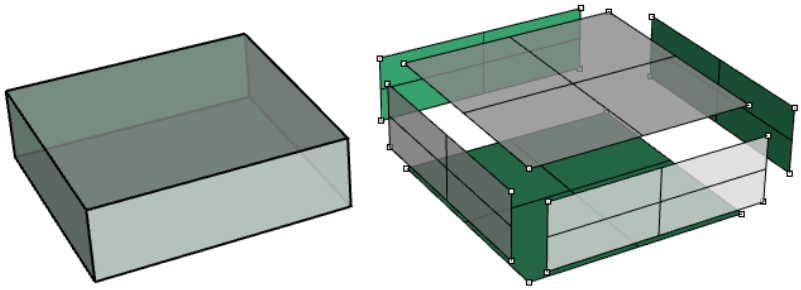


Figura (71): Caja compuesta por seis superficies no recortadas unidas formando una polisuperficie.

Se puede crear la misma caja utilizando superficies recortadas, como la del siguiente ejemplo.

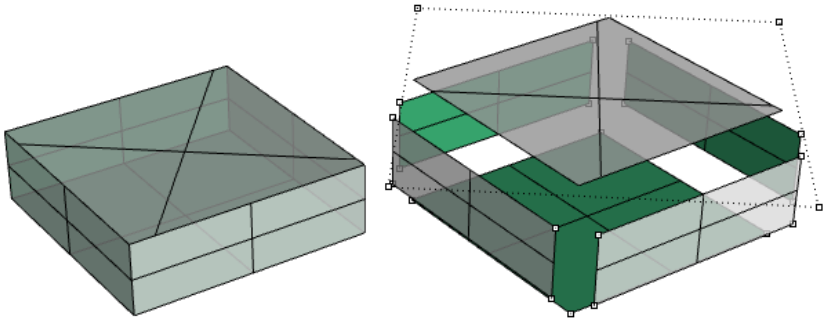


Figura (72): Las caras de la caja pueden ser recortadas.

En el siguiente ejemplo, las caras superior e inferior del cilindro son caras recortadas de superficies planas.

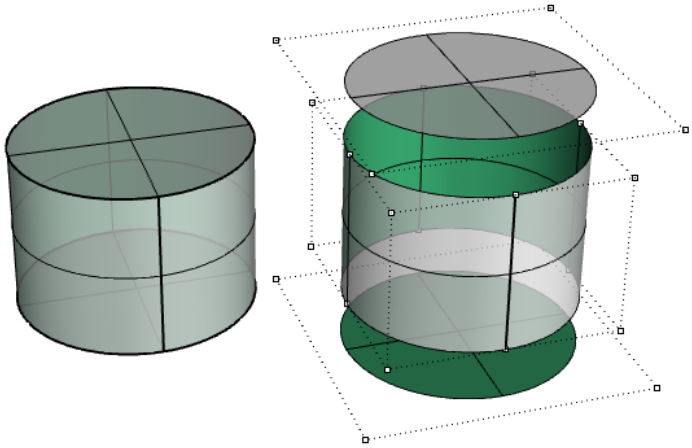


Figura (73): Puntos de control de las superficies subyacentes.

Hemos visto que la edición de curvas NURBS y superficies no recortadas es intuitiva y se puede realizar de forma interactiva moviendo los puntos de control. Sin embargo, la edición de superficies y polisuperficies recortadas no es tan fácil. Lo que resulta complicado es mantener los bordes unidos de las diferentes caras dentro de la tolerancia deseada. Las caras adyacentes que comparten bordes comunes se pueden recortar y no suelen tener una estructura NURBS coincidente, por lo que modificar el objeto deformando ese borde común podría generar una abertura.

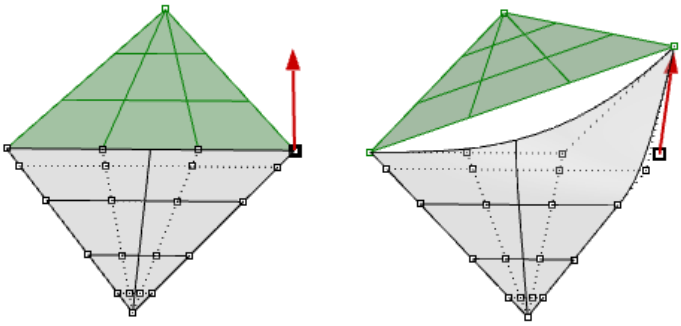


Figura (74): Dos caras triangulares unidas formando una polisuperficie pero sin borde de unión coincidente. Al mover una esquina se crea un agujero.

Otra dificultad es que normalmente hay menos control sobre el resultado, sobre todo cuando se modifica la geometría recortada.

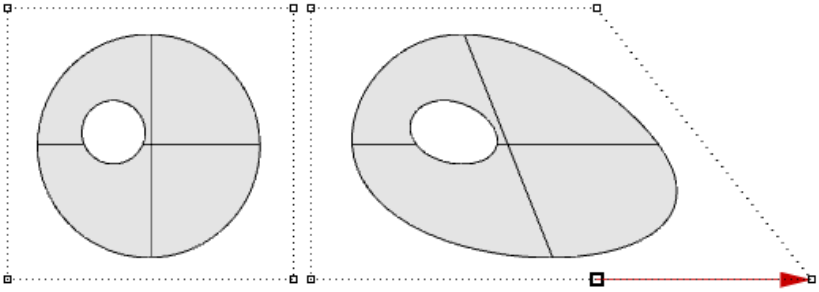


Figura (75): Cuando se crea una superficie recortada, el control para editar el resultado es limitado.

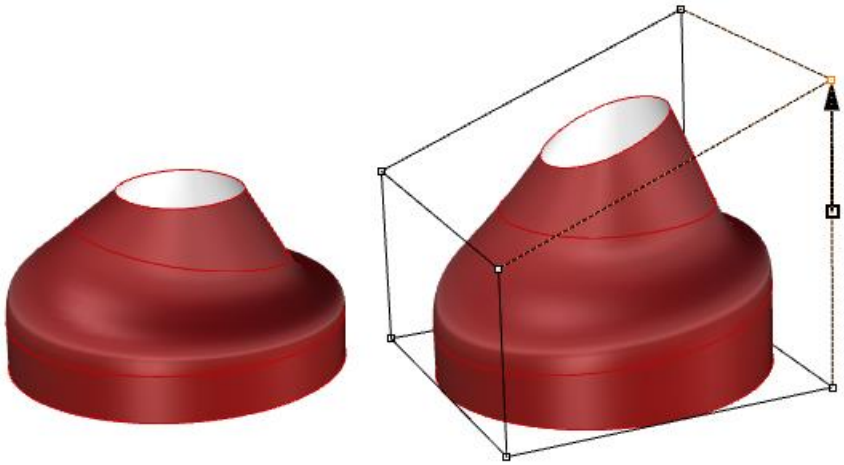


Figura (76): Utilice la técnica de edición de jaula en Rhino para editar polisuperficies.

Las superficies recortadas se describen en el espacio de parámetros utilizando la superficie subyacente no recortada combinada con las curvas de corte 2D que pasan a ser bordes 3D dentro de la superficie 3D.

Tutoriales

Los siguientes tutoriales utilizan los conceptos aprendidos en este capítulo. Utilizan Rhinoceros 5 y Grasshopper 0.9.

Continuidad entre curvas

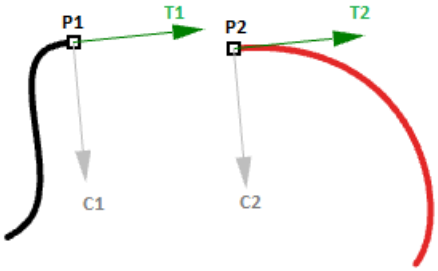
Analiza la continuidad entre dos curvas de entrada. La continuidad supone que las curvas se encuentran al final de la primera curva y al inicio de la segunda curva.

Entrada:

Dos curvas de entrada.

Parámetros:

Calcule los siguientes parámetros para conocer la continuidad entre dos curvas:

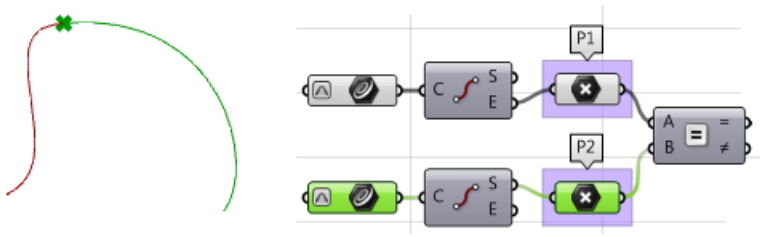


- El punto final de la primera curva (P1).
- El punto inicial de la segunda curva (P2).
- La tangente al final de la primera curva y al inicio de la segunda curva (T1 y T2).
- La curvatura al final de la primera curva y al inicio de la segunda curva (C1 y C2).

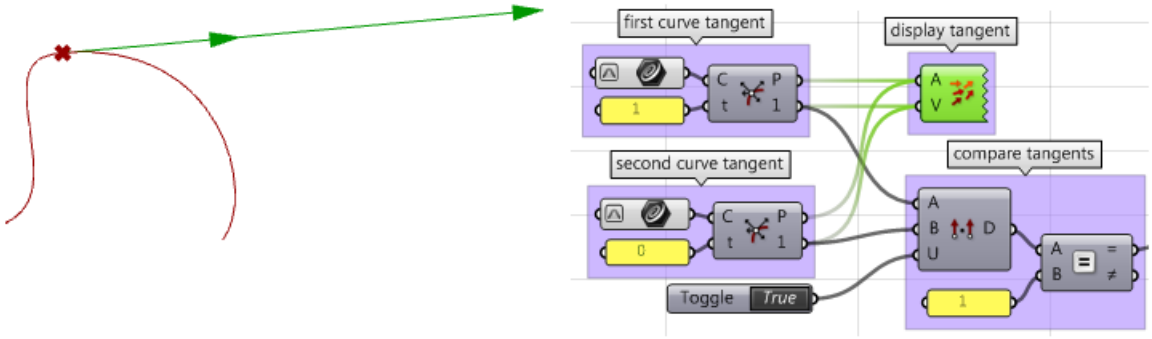
Solución:

1. Reparametrice las curvas de entrada. Lo hacemos para saber que el inicio de la curva se calcula en t=0 y el final en t=1.

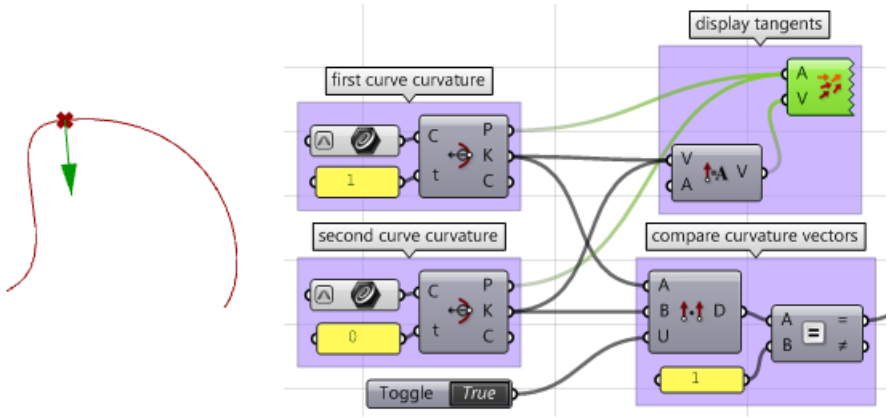
2. Extraiga los puntos final e inicial de las dos curvas y compruebe si coinciden. Si es así, las dos curvas son tienen al menos continuidad G0.



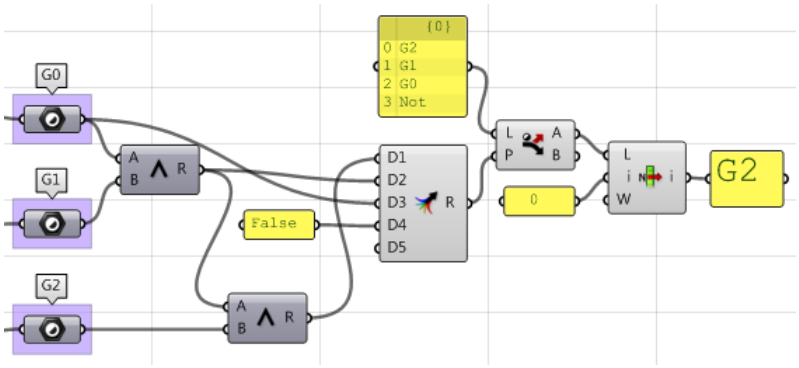
3. Calcule las tangentes.
4. Compare las tangentes utilizando el producto escalar. Asegúrese de unificar los vectores. Si las curvas son paralelas, entonces tenemos al menos continuidad G1.



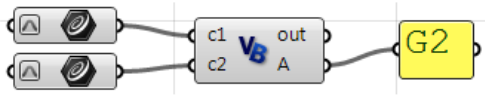
5. Calcule los vectores de curvatura.
6. Compare los vectores de curvatura y, si coinciden, las dos curvas tienen continuidad G2.



7. Cree una lógica que filtre los tres resultados (G1, G2 y G3) y seleccione la continuidad máxima.

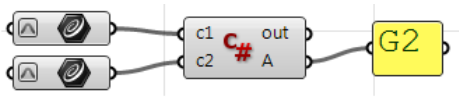

G0
G1
G2
A
B
R
False
{0}
0 G2
1 G1
2 G0
3 Not
D1
D2
D3
D4
D5
L
P
A
B
0
L
i
W
G2

Uso del componente VBScript de Grasshopper:



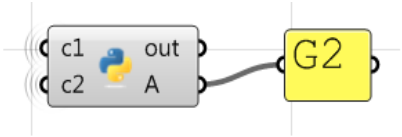

```
Private Sub RunScript(ByVal c1 As Curve, ByVal c2 As Curve, ByRef A As Object)

  'declare variables
  Dim continuity As New String("")
  Dim t1, t2 As Double
  Dim v_c1, v_c2, c_c1, c_c2 As Vector3d

  'extract start and end points
  Dim end_c1 = c1.PointAtEnd
  Dim start_c2 = c2.PointAtStart

  'check G0 continuity
  If end_c1.DistanceTo(start_c2) = 0 Then
    continuity = "G0"
  End If

  'check G1 continuity
  If continuity = "G0" Then
    'calculate tangents
    v_c1 = c1.TangentAtEnd
    v_c2 = c2.TangentAtStart
    'unitize tangent vectors
    v_c1.Unitize
    v_c2.Unitize
    'compare tangents
    If v_c1 * v_c2 = 1 Then
      continuity = "G1"
    End If
  End If

  'check G2 continuity
  If continuity = "G1" Then
    'extract the parameter at start and end of the curves domain
    t1 = c1.Domain.Max
    t2 = c2.Domain.Min
    'calculate curvature
    c_c1 = c1.CurvatureAt(t1)
    c_c2 = c2.CurvatureAt(t2)
    'unitize curvature vectors
    c_c1.Unitize
    c_c2.Unitize
    'compare vectors
    If c_c1 * c_c2 = 1 Then
      continuity = "G2"
    End If
  End If

  'Assign output
  A = continuity
End Sub
```

Uso del componente C# de Grasshopper:




```
private void RunScript(Curve c1, Curve c2, ref object A)
{
  //declare variables
  string continuity = ("");
  double t1, t2;
  Vector3d v_c1, v_c2, c_c1, c_c2;

  //extract start and end points
  Point3d end_c1 = c1.PointAtEnd;
  Point3d start_c2 = c2.PointAtStart;

  //check G0 continuity
  if( end_c1.DistanceTo(start_c2) == 0){
    continuity = "G0";
  }

  //check G1 continuity
  if( continuity == "G0")
  {
    //calculate tangents
    v_c1 = c1.TangentAtEnd;
    v_c2 = c2.TangentAtStart;
    //unitize tangent vectors
    v_c1.Unitize();
    v_c2.Unitize();
    //compare tangents
    if( v_c1 * v_c2 == 1 ){
      continuity = "G1";
    }
  }

  //check G2 continuity
  if( continuity == "G1" )
  {
    //extract the parameter at start and end of the curves domain
    t1 = c1.Domain.Max;
    t2 = c2.Domain.Min;
    //calculate curvature
    c_c1 = c1.CurvatureAt(t1);
    c_c2 = c2.CurvatureAt(t2);
    //unitize curvature vectors
    c_c1.Unitize();
    c_c2.Unitize();
    //compare vectors
    if( c_c1 * c_c2 == 1 ){
      continuity = "G2";
    }
  }

  //assign output
  A = continuity;
}
```

Uso del componente Python de Grasshopper:



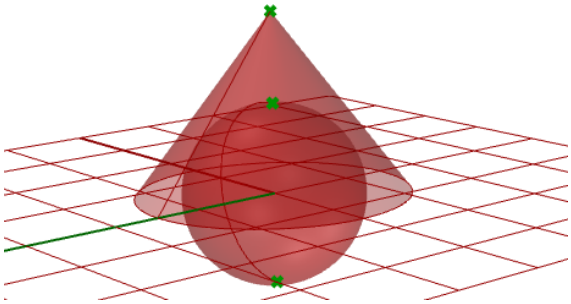

```
#decalre variables
continuity = ""

#extract start and end points
end_c1 = c1.PointAtEnd
start_c2 = c2.PointAtStart

#check G0 continuity
if end_c1.DistanceTo(start_c2) == 0:
    continuity = "G0"

#check G1 continuity
if continuity == "G0":
    #calculate tangents
    v_c1 = c1.TangentAtEnd
    v_c2 = c2.TangentAtStart
    #unitize tangent vectors
    v_c1.Unitize()
    v_c2.Unitize()
    #compare tangents
    dot = v_c1 * v_c2
    if dot == 1:
        continuity = "G1"
        .
#check G2 continuity
if continuity == "G1":
    .
    #extract the parameter at start and end of the curves domain
    t1 = c1.Domain.Max
    t2 = c2.Domain.Min
    #calculate curvature
    c_c1 = c1.CurvatureAt(t1)
    c_c2 = c2.CurvatureAt(t2)
    #unitize curvature vectors
    c_c1.Unitize()
    c_c2.Unitize()
    #compare vectors
    dot = c_c1 * c_c2
    if dot == 1:
        continuity = "G2"
        .
#assign output
A = continuity
```

Superficies con singularidad

Extraiga puntos singulares en una esfera y un cono.

Entrada:

Una esfera y un cono.



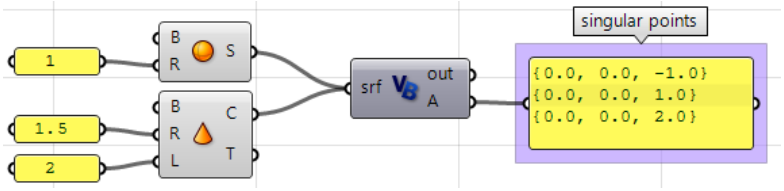
Parámetros:

La singularidad se puede detectar analizando los recortes del espacio de parámetros 2D que tienen bordes correspondientes no válidos o de longitud cero. Esos recortes deberían ser singulares.

Solución:

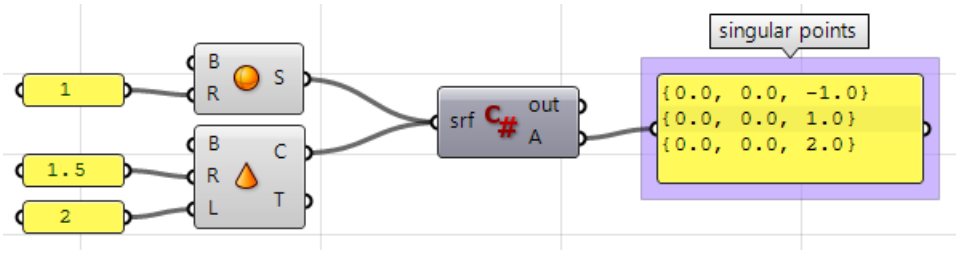
1. Compruebe todos los recortes de la entrada.
2. Compruebe si algún recorte tiene un borde no válido y márquelo como recorte singular.
3. Extraiga las ubicaciones de los puntos en el espacio 3D.

Uso del componente VB de Grasshopper:



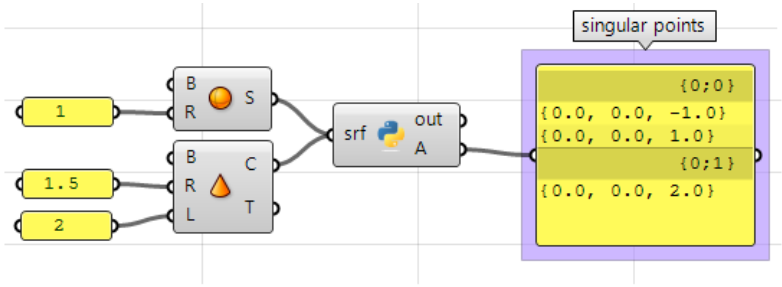
```
Private Sub RunScript(ByVal srf As Brep, ByRef A As Object)

  'Decalre a new list of points
  Dim singular_points As New List( Of Point3d)

  'Examine all trims in the input
  For Each trim As BrepTrim In srf.Trims

    'Null edge of a trim indicates a singularity
    If trim.Edge Is Nothing Then

      'Find the 2D parameter space point of the start or end of the trim
      Dim pt2d = New Point2d(trim.PointAtStart)

      'Evaluate trim end point on the object surface
      Dim pt3d = trim.Face.PointAt(pt2d.x, pt2d.y)

      'Add 3D point to the list of singular points
      singular_points.Add(pt3d)
    End If

  Next

  'Asign output
  A = singular_points

End Sub
```

Uso del componente C# de Grasshopper:



```
private void RunScript(Brep srf, ref object A)
{
  //Decalre a new list of points
  List < Point3d > singular_points = new List<Point3d>();

  //Examine all trims in the input
  foreach( BrepTrim trim in srf.Trims)
  {
    //Null edge of a trim indicates a singularity
    if( trim.Edge == null)
    {
      //Find the 2D parameter space point of the start or end of the trim
      Point2d pt2d = new Point2d(trim.PointAtStart);

      //Evaluate trim end point on the object surface
      Point3d pt3d = trim.Face.PointAt(pt2d.X, pt2d.Y);

      //Add 3D point to the list of singular points
      singular_points.Add(pt3d);
    }
  }

  //Asign output
  A = singular_points;
}
```

Uso del componente Python de Grasshopper:




```
#Decalre a new list of points
singular_points = []

#Examine all trims in the input brep
for trim in srf.Trims:

    #Null edge of a trim indicates a singularity
    if trim.Edge == None:
        #Find the 2D parameter space point at trim start or end
        pt2d = trim.PointAtStart

        #Evaluate trim end point on the object surface
        pt3d = trim.Face.PointAt(pt2d.X, pt2d.Y)

        #Add 3D point to the list of singular points
        singular_points.append(pt3d)

#Asign output
A = singular_points
```

Referencias

Edward Angel, "Interactive Computer Graphics with OpenGL", Addison Wesley Longman, Inc., 2000.

James D Foley, Steven K Feiner, John F Hughes, "Introduction to Computer Graphics", Addison-Wesley Publishing Company, Inc., 1997.

James Stewart, "Calculus," Wadsworth, Inc., 1991.

Kenneth Hoffman, Ray Kunze, "Linear Algebra", Prentice-Hall, Inc., 1971

Ayuda de Rhinoceros®, Robert McNeel and Associates, 2009.

Notas